Semi-automatic Extraction of Primitive Geometric Entities from Point Clouds

Charl Leonard Goussard

Thesis presented in partial fulfillment of the requirements for the degree of Master of Science in Engineering (Mechanical) at the University of Stellenbosch

Semi-automatic Extraction of Primitive Geometric Entities from Point Clouds

Charl Leonard Goussard

Thesis presented in partial fulfillment of the requirements for the degree of Master of Science in Engineering (Mechanical) at the University of Stellenbosch

Thesis Supervisor: Prof. A.H. Basson
Department of Mechanical Engineering
University of Stellenbosch
December 2001

Declaration

I, the undersigned, hereby declare that the work contained in this thesis is my own original work and that I have not previously in its entirety or in part submitted it at any university for a degree.

Abstract

This thesis describes an algorithm to extract primitive geometric entities (flat planes, spheres or cylinders, as determined by the user's inputs) from unstructured, unsegmented point clouds. The algorithm extracts whole entities or only parts thereof. The entity boundaries are computed automatically. Minimal user interaction is required to extract these entities. The algorithm is accurate and robust.

The algorithm is intended for use in the reverse engineering environment. Point clouds created in this environment typically have normal error distributions. Comprehensive testing and results are shown as well as the algorithm's usefulness in the reverse engineering environment.

Opsomming

Hierdie tesis beskryf 'n algoritme wat primitiewe geometriese entiteite (plat vlakke, sfere of silinders na gelang van die gebruiker se inset) pas op ongestruktureerde, ongesegmenteerde puntewolke. Die algoritme pas geslote geometriese entiteite of slegs dele daarvan. Die grense van hierdie entiteite word automaties bereken. Minimale gebruikersinteraksie word benodig om die geometriese entiteite te pas. Die algoritme is akkuraat en robuust.

Die algoritme is ontwikkel vir gebruik in die truwaartse ingenieurswese omgewing. Puntewolke opgemeet in hierdie omgewing het tipies meetfoute met 'n normaal verdeling. Omvattende toetsing en resultate word getoon en daarmee ook die nut wat die algoritme vir die gebruiksomgewing inhou.

Dedication

This thesis is dedicated to my parents for their encouragement and support during the past few years.

Acknowledgements

The author wishes to express his gratitude towards the following individuals and institutions for contributing towards this study:

- The NRF, Prof. A.H. Basson and my parents for financial assistance.

- Prof. A.H. Basson, my thesis supervisor, for his valuable advice and criticisms during the research for this thesis.

- My fellow student, Kristiaan Schreve, for his unselfish guidance, reliable support and advice.

- Finally, my Heavenly Father, without whom nothing is possible and to whom all credit is due.

Table of Contents

List of Figures

List of Tables

Nomenclature

Bold uppercase	Matrix
Bold lowercase	Vector

a	Direction cosine of line with respect to the x-axis
b	Direction cosine of line with respect to the y-axis
c	Direction cosine of line with respect to the z-axis
d	Distance
E	Function
F	Function
$\mathbf{p}$	Vector of update values
r	Radius
x	Cartesian coordinate
y	Cartesian coordinate
z	Cartesian coordinate

Greek Symbols

α	Scalar
δ	Vector of distances from each point to the entity
∂	Partial differentiation
$\delta_{i,j}$	Kronecker δ : 0 if i ≠j; 1 if i=j
λ	Eigenvalue
θ	Angle

Superscripts

$'$	First transformation or rotation
$''$	Second transformation or rotation
T	Transpose
k	Current iteration value
$k+1$	Next iteration value

Subscripts

i	Row index
m	Number of rows
n	Number of columns
o	Entity parameter value

Auxiliary Symbols

–	Average
| |	Determinant of matrix

Glossary

2D	Two Dimensional
3D	Three Dimensional
CAD	Computer Aided Design
CMM	Coordinate Measuring Machine
NURBS	Non-Uniform Rational B-Spline
SVD	Singular Value Decomposition

1 Introduction

Reverse engineering involves constructing a CAD model, generally composed of curves and surfaces, from a part or a prototype. Laser scanners and coordinate measuring machines (CMM) are commonly used to generate clouds of points representing curves and surfaces on the object. These points are then used to construct a best-fit surface to the data using curve and surface fitting techniques. This procedure may be used to create CAD models from manufactured parts when the drawings are not available for a part, or to modify an existing CAD model after the part has been manufactured as in the case of the die and mould industry for instance. The main aim of this thesis is to reduce the reverse engineering time by recognizing primitive manufactured features such as drilled holes or milled surfaces associated with the part.

To fit entities accurately and robustly, it is necessary to find appropriate initial parameters for different entities. A sphere, for example, has four parameters, i.e. its centre point (x, y and z coordinates) and its radius. Methods to find these initial parameters and to fit entities once the initial parameters are known, will be discussed.

Computer algorithms to find the borders of entities will be discussed in detail, as this is not a trivial task. Test cases are examined to provide information about the fitting processes of different entities.

Finally some remarks will be given about the accuracy of the computer algorithms and the conclusions of the research will be stated. In this chapter the research will be motivated.

1.1 Reverse Engineering

In reverse engineering real parts are transformed into engineering models and concepts as opposed to transforming engineering concepts and models into real parts in conventional engineering. The reverse engineering process starts with measuring

the surfaces of an object. This is accomplished with the use of coordinate measuring machines or laser scanners for instance. A point cloud that contains all the three-dimensional points scanned from the object's surfaces is created. The next step usually is to cut the point cloud along surface boundaries to form different segments of the object. Free-form surfaces (NURBS surfaces for example) or geometric entities (employing quadric surfaces for example) are then fitted to these segments. The surfaces are then trimmed or extended to find the complete surface of the measured object. CAD drawings of the object and cutter paths for the manufacturing of the object can then be obtained.

Metwalli et al. [1999], Várady et al. [1997] and Chivate and Jablokow [1995] are of the many researchers who state why reverse engineering is necessary. It is necessary to create a CAD model for an existing part or prototype for the following cases:

- A new design is created by modifying an existing design for which no CAD model exists.
- Worn out parts, parts that have failed or one of a kind parts have to be reproduced.
- A physical prototype was built from preliminary design ideas and therefore no or few drawings of the part exist.
- The CAD database has not been updated according to continual changes in the product design.
- Aesthetic design is particularly important in the automobile industry. Therefore real-scale wood or clay models are needed because stylists often rely more on evaluating real 3D objects than on viewing projections of objects on high resolution 2D screens at reduced scale. These real 3D models must be reverse engineered to obtain CAD models.
- Manufactured parts are measured and the produced CAD models are compared to the original drawings for inspection purposes. This is the only case where original CAD drawings exist for the reverse engineered parts.

1.2 Research Objectives

The main objective of this thesis is to develop a technique to fit primitive geometric entities (planes, spheres and cylinders) accurately to a provided point cloud as

generated in the reverse engineering field. This fitting process must be done without prior segmentation of the point cloud and the boundaries of the primitive geometric entities must be found automatically. Minimal user interaction must be required for the identification of selected entities. Figure 1.1 shows where the research fits into the field of bounding surfaces.

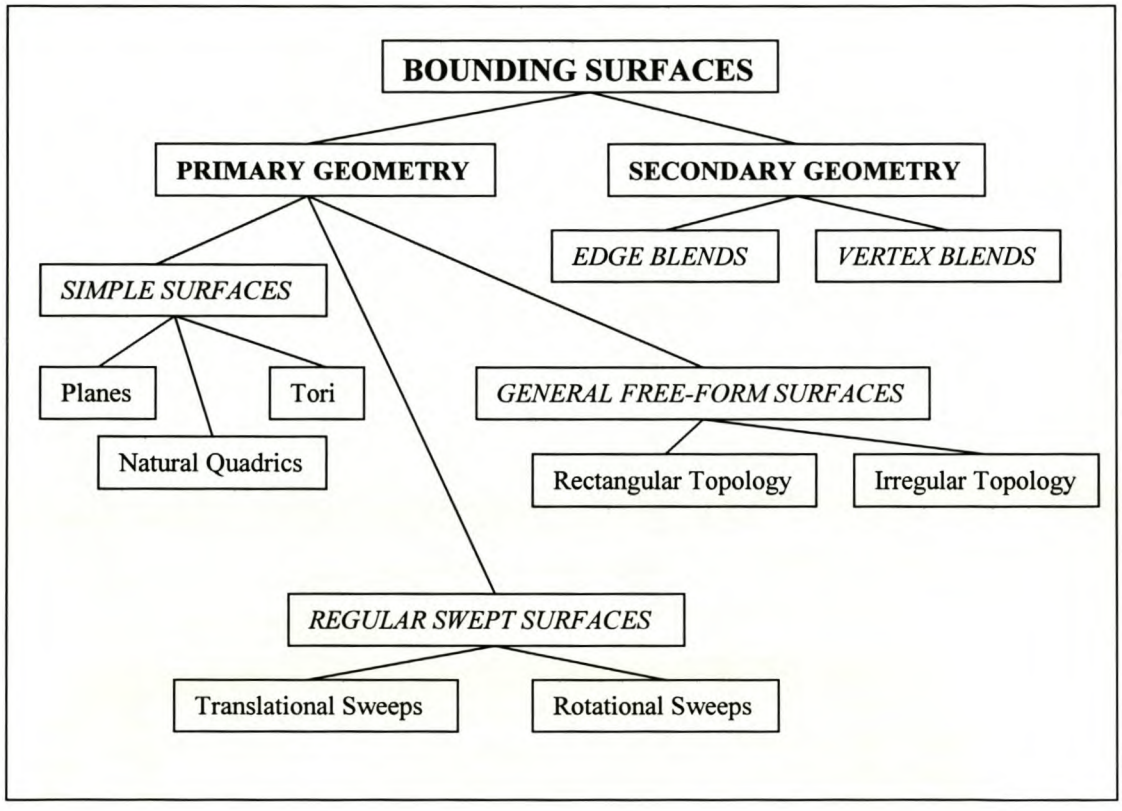

Figure 1.1 Hierarchy of surfaces

1.3 Research Motivation

Schreve and Basson [2000] explain the difficulties arising in manually identifying surface boundaries when fitting NURBS surfaces. It is difficult to distinguish whether points lying close to the boundary of an entity actually belong to that entity or to its neighbour entity. These boundary points can therefore not be included in the surface NURBS fit and therefore surfaces must be extended to find the real boundary.

Skilled designers are needed to reconstruct parts using free form fitting methods. Fitting the different patches together is not a trivial task and is time consuming [Park and Jun, 2000]. Extending and trimming of surfaces is necessary to join the different patches. Extending surfaces, especially cubic (or higher degree) NURBS surfaces, often leads to an undesirable shape of the surface. Natural quadrics have the advantage to be easily extended or trimmed in order to find the real boundaries. Therefore design changes are more easily implemented when geometric entities are used.

Primitive entity recognition can help to identify the simplest machining operations (for example milling and drilling) necessary to manufacture the original design. This will lead to simpler manufacturing procedures and lower manufacturing cost.

Robust sharing of simple geometric entities between different modelling packages is possible. Geometric entities have simple equations compared to polynomial free-form surfaces. A large amount of information has to be conveyed between modelling packages, especially for cases where free-form surfaces are trimmed or extended. Therefore the chances of interpreting modelling information incorrectly improve when using free-form surfaces.

Measured data, for example from a coordinate measuring machine, contains a certain amount of measurement error or noise. Free-form surface fitting techniques may result in irregular surfaces if the point data have a large amount of noise, unless smoothing is introduced. Smoothing the data may, however, introduce bigger errors. It is necessary to get rid of these noisy surfaces to be able to manufacture the reverse engineered part, without decreasing accuracy too much. Identifying geometric primitives associated with the part can speed up this process.

In summary, using primitive geometric entities can decrease the time spent on cutting surface segments, finding borders of entities, filtering data and smoothing data.

1.4 Related Research Projects

The current research project runs in parallel with that of Schreve and Basson's [2000]. They are developing a method that reconstructs edges (sharp boundaries) from point clouds. The edges are found by simulating the movements of a CMM. For this purpose a virtual CMM is being developed. The edge detection method is semi-automatic and uses similar input to that required for scanning a contour on a CMM. This technique can be implemented on a real CMM as well.

1.5 Thesis Overview

The need for developing a tool to extract simple geometric entities from point clouds in the reverse engineering environment and the goals of the research were presented in this chapter. Chapter 2 focuses on the literature review. Data acquisition methods, implications of using free-form surfaces, different data structures and mathematical methods to model geometric entities are discussed.

Chapter 3 describes methods to fit geometric primitive entities and Chapter 4 provides information about data structures and neighbour search routines. The octree data structure is discussed in detail.

Chapter 5 focuses on the computer algorithm written to extract the geometric entities and Chapter 6 discusses the validation of the algorithm with different test cases. The conclusion and closure are presented in Chapter 7 and references are given in Chapter 8.

2 Literature Review

This chapter starts with literature on primitive entities and how it can be used in practise. Data acquisition and possible related errors are discussed thereafter. The implications of free-form surface fitting and difficulties arising in entity boundary approximation are reviewed. The review follows this with a description of data structures used in practise. The literature review closes with different mathematical models used to fit surfaces and entities.

2.1 The Usefulness of Geometrical Entities

Even in today's modern manufacturing environment, simple geometric features are still part of mechanical parts. Many parts have free form surfaces combined with simple milled surfaces and drilled holes. Simple geometric features can be mathematically formulated by natural quadrics. Hakala et al. [1980] reported that natural quadrics represent 85% of machined objects. The natural quadric equation has ten coefficients and assigning correct values to these coefficients can represent a specific geometric primitive.

Geometrical constraints can be formulated if natural quadrics are used to represent simple geometric entities. A cylinder axis has to be perpendicular to a plane for example. The cylinder and the plane each have its own coefficients for the natural quadric equation. This geometrical constraint can be achieved by applying relationships between these two quadric equation's coefficients. This cannot be done if the object was modelled with free form surfaces. Research done by Werghi et al. [1999] shows the different relationships between the coefficients to obtain certain geometrical constraints.

Park and Jun [2000] proposed a new methodology of automatically reconstructing analytic surfaces such as planes, cylinders, spheres and cones from scanned data. Triangular meshes are generated from point data in order to find the boundaries. They also mention that fitting free-form surfaces is a non-trivial time-consuming task.

2.2 Data Acquisition and Possible Errors

Many different methods exist for acquiring shape data as shown in Figure 2.1.

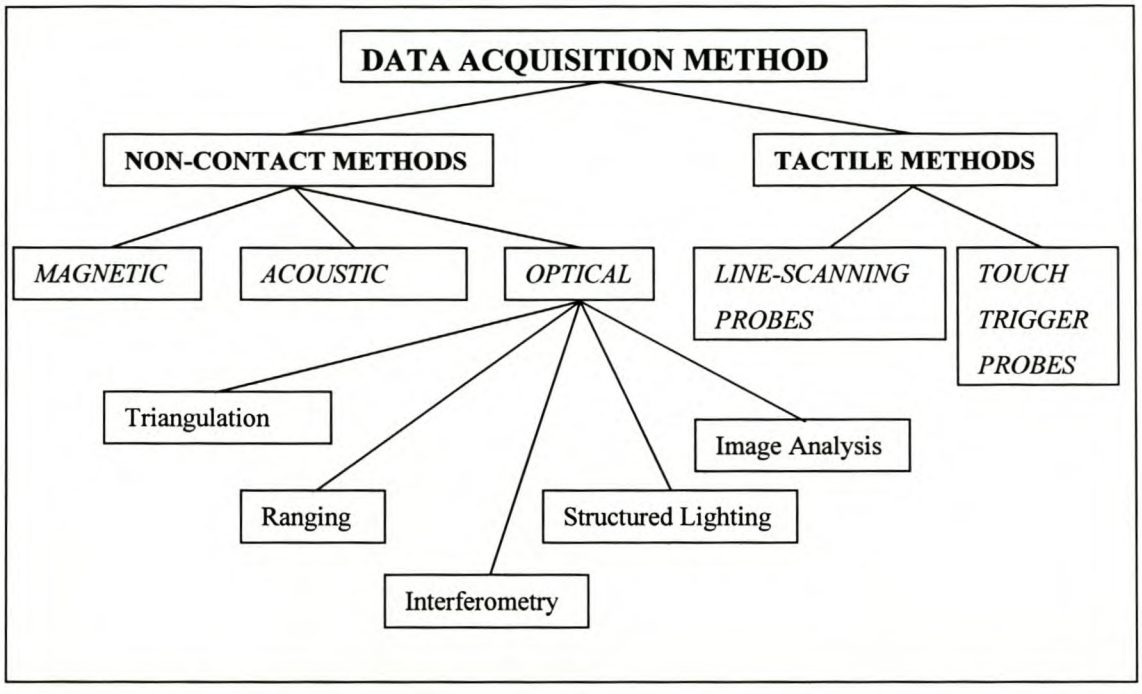

Figure 2.1 Classification of data acquisition methods

Optical methods include triangulation, ranging, interferometry, structured lighting and image analysis. Optical methods of shape capture are probably the broadest and most popular with relatively fast acquisition rates according to Várady et al. [1997]. Triangulation methods use a high-energy light source (a laser beam for instance) to capture data. The light source is aimed at a specific angle to a surface. A photosensitive device, usually a video camera, captures the reflection off the surface. A point on the surface is calculated by using geometry of the known angles and distances.

Ranging methods use laser and pulsed beams. The light beam is projected to the surface and the reflection off the surface is captured. Points on a surface are calculated by sensing the time-of-flight of the light beams. Interferometry methods measure distances in terms of wavelengths using interference patterns. Very accurate measurements can be obtained with this method since the wavelength of visible light for instance is in the order of 0.4 - 0.7 μm [Mills, 1992].

The structural lighting technique involves projecting patterns of light onto a surface and the reflection pattern is captured as an image. Shadow Moiré is a structured lighting technique where contour lines are generated on a surface by projecting an interference pattern on the surface. These contour lines are analysed to determine the distance between the lines. This distance is proportional to the height of the surface at a point of interest and the coordinates of this point can be calculated. A single image frame is used to produce many data points, but to determine the positions of the points accurately is difficult. Image analysis differs from structural lighting as stereo pairs are used to determine height and position of the points as opposed to using projected patterns to do the calculations.

Tactile methods require the use of mechanical arms. A probe is fitted to the machine and if the probe touches an object, sensing devices in the joints of the arm provides the position of the touched point. A 3-axis milling machine can be used as the mechanical arm. Tactile methods are very robust in terms of low noise, accuracy and repeatability but have the drawback of being the slowest method for data acquisition. Coordinate measuring machines (CMM) can be programmed to measure along paths on a surface to collect very accurate data. Chivate and Jablokow [1993] generated data with the use of a CMM and describes in detail how the solid model was produced. Van Vliet and Schellekens [1996] describe how increasing the measuring speed of CMM's affects the accuracy of the measurements.

The time-of-flight of sound is used to determine distance in acoustic methods. The sound source is reflected off the surface and with the known speed of sound it is easy to determine the distance between the source and the surface. This method is vulnerable to acoustic interference or other noise. Magnetic field measurement involves sensing the strength of a magnetic field source. Magnetic touch probes are used which usually sense the location and orientation of a stylus within the field. The strength of the magnetic field is used to calculate the location of a specific point.

Várady et al. [1997] and Milroy et al. [1997] describe different causes of errors that occur during data acquisition. Accurate calibration and determination of accuracy are important with any data acquisition method. Any sensing devices must be calibrated to accurately determine parameters such as camera points and orientations, and to

model and allow for systematic sources of error as accurately as possible. Inaccessible surfaces like through holes require the use of multiple scans, which may cause inaccurate results when these views are combined. Multiple views are often related by fixing a reference sphere on the part that is visible in all views. Data are shifted so that the centre of the reference sphere appears to coincide in each of the views. In some cases certain methods of data acquisition are impossible. Occlusion occurs when the scanning medium is blocked by a shadow (relevant to optical scanners) or other obstructions. The part itself can create this problem, as well as fixturing devices, which are used to clamp the part to a table or platform while data acquisition is being performed.

Noise is created during any data acquisition method. The data is corrected by noise filtering which often destroys the 'sharpness' of the data, e.g. sharp entity boundaries that are replaced with smooth blends. Missing data due to inaccessibility or obstruction must be restored in some way (surface extensions, intersections, patching holes etc.). Data close to sharp boundaries is often unreliable and surfaces must be extended to find the true boundary. Surface finish produces further practical problems. More noise is generated when measuring with a tactile probe on a rough surface than on a smooth surface. Reflective coatings can introduce noise when measuring with an optical scanner.

2.3 Implications of Using Free-Form Surfaces

Free-form surfaces, especially NURBS, are commonly used to model mechanical parts. NURBS are used in particular because it has the ability to model free-form surfaces flexibly and simple quadric surfaces exactly [Lee and Fang, 2000]. This exact representation of simple shapes requires careful and often difficult selection [Sapidis and Besl, 1995] of a number of parameters: the position of the control points, the value of the weights and the knot vector. Piegl and Tiller [1997] provides in depth information regarding NURBS.

NURBS surfaces provide difficulties when they are extended. Extension is often necessary because it is difficult to detect entity boundaries accurately. Finding intersections between NURBS surfaces are not a trivial task either. It is difficult to

model sharp boundaries because fitting NURBS surfaces to point data results in smoothing the surface to relate to the data. Free-form surface fitting requires user experience and is time consuming in summary.

2.4 Difficulties Arising in Boundary Approximation

Milroy et al. [1997] mentions the difficulties regarding finding boundaries of features existing in point clouds. Free-form surfaces have to be fitted to segmented data. The segmentation is a manual, interactive process that has to be carried out on the disorganized data points. The operator has to segment the data in order to hide extraneous data until the remaining points lies on a single valued surface. The computer mouse is then used to trace boundary curves on the data. The precise location of the 3D boundaries is difficult to determine on a flat computer screen, making this a laborious, error-prone process.

Segmenting the point cloud is not a trivial task because all the surface patches have to be joined later and continuity conditions have to be met at the surface boundaries. The designer has to make sure that points not belonging to a certain surface are not included. This is the reason that in most cases the surfaces are not fitted to the actual boundaries [Schreve and Basson, 2000]. Park and Jun [2000] states that this method of cutting patches and fitting free-form surfaces to the patches is time consuming and labour expensive.

2.5 Data Structures

A text file containing the x, y and z coordinates of the measured data points is typically created during the data acquisition phase. The number of data points can easily reach one million and these points are often non-structured. The data acquisition method used to generate the point cloud will strongly influence the size of the point cloud. Generally, non-contact methods create larger point clouds than contact methods. These non-contact generated point clouds are often filtered afterwards to eliminate redundant points. This necessitates the use of an efficient and elegant manner of working with such large data files. It is necessary to have some 3D

information about the points to effectively search for entities in point clouds. Such 3D information can be obtained by constructing Delaunay triangles or octrees.
The Delaunay triangulation scheme works on the basis that a circle drawn through the three points of any triangle will never enclose another point of another Delaunay triangle. Delaunay triangulations were used by Ramakrishnan et al. [1992] to construct finite element grids and by Park and Jun [2000] to extract primitive analytic surfaces.

This research has to be integrated with that of Schreve and Basson's [2000] and therefore the octree technique was chosen. Other reasons include the robustness of the octree technique and the relative little amount of time spent to build the octree data structure. The octree technique uses a method of placing the points in nodes and dividing these nodes until certain stop criteria are met. Therefore points can easily be found in the data set. This method is explained in detail in Chapter 4. This choice created another challenge, as point neighbour information is not conveyed by the octree scheme. Neighbour information is needed to identify points belonging to the same entity. A neighbour search algorithm was developed by Vörös [2000]. Octree techniques were used by Shephard and Georges [1991] and Perucchio et al. [1989] to automatically generate three-dimensional meshes. Yu et al. [1996] used an octree algorithm to detect dynamic interference between mechanical parts. The octree algorithm is explained in more detail by Meagher [1982] and advantages of this geometric modelling technique are given.

2.6 Mathematical Methods

A mathematical model is needed to extract a certain entity after an octree was created from the point cloud. Genetic algorithms, probability methods and general least squares with a Gauss-Newton algorithm are commonly used to solve these problems. Methods exist for estimating entity parameters that are used as good starting values for the different models. Gauss-Newton methods are highly dependent on good initial estimates.

2.6.1 Genetic Algorithms

Genetic algorithms are based on the principles of natural genetics and natural selection. The basic elements of natural genetics – reproduction, crossover and mutation – are used in the genetic search procedure. Genetic algorithms do not start with a single design point. Normally a population of design points of the order of two to four times the number of design variables is used. This is the reason why genetic algorithms are less likely to be trapped at a local optimum. Design variables are represented as strings of binary variables that correspond to the chromosomes in natural genetics. Discrete and integer programming problems are usually solved with genetic algorithms, but solving for continuous design variables are also possible by varying the string length of the design variables. The objective function value corresponding to a design vector plays the role of fitness in natural genetics. A new set of strings is produced in every new generation by using randomised parents selection and crossover from the old generation. Genetic algorithms explore new generation combinations with the available knowledge to find a new generation with better fitness or objective function value.

Genetic algorithms are well explained by Rao [1996]. Limaiem et al. [1996] used a dual kriging method combined with a genetic algorithm to fit curves and surfaces to scanned data. Dual kriging incorporates several interpolation techniques such as piecewise interpolation, cubic splines, Bezier, B-splines and NURBS curves, surfaces and solids in a single formulation. Limaiem et al. [1996] also stressed that the genetic algorithm ensured that their solution is a global optimum. Nassef et al. [1999] compared the Nelder Meade Simplex method (a variant of the traditional direct search method) to a genetic algorithm in fitting a cylinder to data generated by a computer. Their conclusion was that the genetic algorithm fitted the entity more accurately and faster (when minimizing the number of objective function evaluations) than the direct search method. Minimizing the number of objective function evaluations is done by optimising the genetic algorithms parameters such as the number of times for which the various crossovers and mutations should be applied. However, the authors used starting values far from the global optimum for the direct search method. The direct search method may outperform the genetic algorithm if starting values near the global optimum are used.

2.6.2 Probability Methods

Quadric surfaces can be fitted to random cloud data with the use of a probability method based on maximum likelihood estimation. The objective of this statistical approach is to estimate the parameters of a general quadric that most probably correspond to the data points belonging to the specific entity which is part of the whole set of data points. Points belonging to a specific entity are considered as random variables with the entity parameters considered as the mean value function. The entity is fitted to the data by adjusting the quadric surface parameters and orientation in space to minimize an error metric between the selected cloud data points and the quadric surface. Bradley et al. [1994] and Milroy et al. [1996] used probability methods to fit surfaces to point data. They created an interactive 3D graphical program called LaserCAM for surface reconstruction. The 1996 article presents a case study where a water timer housing is reverse engineered.

2.6.3 General Least Squares

2.6.3.1 Reasons for using least squares

Least squares fitting of entities to data is a very efficient method to solve overdetermined sets of linear equations (i.e. when more equations exist than the number of unknown variables). Least squares, as an approximation technique, is ideal for fitting entities when a large number of points with the existence of noise are considered. Least squares filter the noise to allow for a high quality fit [Limaiem et al., 1996]. Least square methods are widely used, as they are simple to compute, easy to implement in computer processing and give a unique solution [Samuel and Shunmugam, 1999]. The work presented in this thesis concentrates on the use of least square fitting methods.

Fitting entities by the means of formulating a least squares problem is widely reported in literature [Taubin, 1991 and Thompson et al., 1999]. Thompson et al. [1999] has written a program REFAB that uses manufacturing features as geometric primitives.

Least squares fitting, least absolute fitting and least median of squares fitting were used to fit geometric entities to point data by Chivate and Jablokow [1995]. They

concluded that least median of squares fitting is the best approximation scheme because it is not as sensitive to outliers (points that do not belong to the geometric entity). However, computational time increases dramatically as the number of data points increases when using this method.

Forbes [1991] presented least squares algorithms to fit certain geometric entities. The parameterisation, distance from a point to the entity surface and algorithm description are given for lines, planes, circles, spheres, cylinders and cones. This paper focuses on data points that are reasonably accurate and representative of the geometric element to be fitted. Forbes [1989] preceded this paper with one where the point data were not representing the whole entity. He used least squares to fit circles and spheres to point data where the data only covered a small patch of the entity surface.

2.6.3.2 Formulation of the least squares problem

The following steps are necessary to fit an entity with the use of least squares [Forbes, 1991]: Parameters must be identified which describe the position, orientation, size and shape of the geometric entity. An expression has to be formulated for the distance from a point to the entity surface in terms of these parameters. The objective of least squares is to find the parameters that will make the sum of distances squared a minimum.

Singular value decomposition can be used to solve the linear least squares problem [Lawson and Hanson, 1995]. It can be directly applied to fit planes because the expression for the distance from a point to a plane surface is linear (Section 3.1). The distance equation of spheres and cylinders are non-linear (Sections 3.2 and 3.3). A Gauss-Newton algorithm can be used to linearise the set of non-linear equations so that a linear least squares system can be obtained to solve for the unknown entity parameter values. The Gauss-Newton algorithm requires good initial estimates (Section 2.6.4) of the unknown entity parameter values to obtain an accurate solution. Cox and Jones [1989] used a Gauss-Newton algorithm to solve for the parameters of a least squares circle. They made use of the approximate least squares circle, which is a linear problem, to estimate the centre coordinates and radius.

2.6.4 Initial Estimation of Entity Parameter Values

The quadric equation can be used [Chivate and Jablokow, 1993, 1995 and Werghi et al., 1999] to obtain initial parameter information for different entities. At least ten points are necessary to solve for the initial parameters of the general quadric equation as it contains ten coefficients. Different relations between the ten parameters exist for each entity. These relations are described by Werghi et al. [1999]. Least squares are used to solve the values of the ten coefficients. Reliable solutions are obtained when the data points used for the solution are representative of the geometric element concerned. However, poor estimates are obtained if small patches of data points are used on a specific entity surface.

3 Fitting of Geometric Entities

This chapter gives general fitting information for each entity considered when the points that belong to the entity are already known. Parameters that describe the position, orientation, size and shape of an entity are found for each entity considered. A formula is then derived for the distance of a point to the entity surface in terms of these parameters. In the case of a flat plane, this equation will be linear. This equation will be non-linear in the case of spheres and cylinders. Methods for finding initial estimates of the entity parameters are necessary if the distance equation is non-linear. Linear least squares (and a Gauss-Newton algorithm if necessary) is/(are) used to solve the entity parameter values. These two concepts are explained in Sections 3.1 and 3.2.

3.1 Linear Least Squares

Linear least squares [Forbes, 1991] is used to fit the three different entities to the included data points belonging to the selected nodes.

The fitting process finds the minimum value of

$$E(\mathbf{u}) = \sum_{i=1}^{m} d_i^2(\mathbf{u}) \tag{3.1}$$

with respect to the parameters:

$$\mathbf{u} = (u_1, u_2, ..., u_n)^T \tag{3.2}$$

$E(\mathbf{u})$ is the sum of all distances d_i (Sections 3.3 - 3.5) squared (for m data points) in terms of the entity parameter values $\mathbf{u}$. An equation

$$\mathbf{Au} = \mathbf{b} \tag{3.3}$$

can be created in the case of linear distance equations or after linearising a set of non-linear distance equations with a Gauss-Newton algorithm. In general, much more data points (m) will exist than unknown entity parameter values ($u_1, \ldots, u_n$). Since $m > n$, all the equations of (3.3) cannot be solved simultaneously and therefore the least squares solution, which minimises $E(\mathbf{u})$, is used.

The solution of the parameters **u** can be found by computing the singular value decomposition (SVD) of the matrix **A**,

$$\mathbf{A} = \mathbf{U}\mathbf{S}\mathbf{V}^T \tag{3.4}$$

where **U** and **V** are orthogonal matrices and therefore their inverses are equal to their respective transpose matrices. **S** is a diagonal matrix and therefore its inverse is the diagonal matrix whose elements are the reciprocals of the elements of **S**. Therefore the solution of **u** is:

$$\mathbf{u} = \mathbf{V}\left[diag\left(1/s_j\right)\right]\mathbf{U}^T\mathbf{b} \tag{3.5}$$

where s_j are the elements of **S**. All numbers given by $1/s_j$ are replaced with zero, if any s_j are zero or almost zero compared to the other s_j . This method is proposed by Press et al. [1997] to counteract the influences of round off problems that can occur if the matrix **A** is near singular.

Other methods exist to solve equation 3.3, for instance Givens transformations and QR factorisation [Golub and van Loan, 1993]. Singular value decomposition was chosen for this thesis for its robustness and stability [SVD code from Press et al., 1997] to solve equation 3.3 in the least squares sense. Press et al. [1997] recommend singular value decomposition for all but "easy" least squares problems. This solving method can be expressed as solving the roots of the following equation:

$$\mathbf{A}\mathbf{u} - \mathbf{b} = \mathbf{0} \tag{3.6}$$

3.2 Gauss-Newton Algorithm

Figure 3.1 shows the principle of the Gauss-Newton method [Forbes, 1991 and Rao, 1996] for a single equation. The first estimate ($u_0(g)$) of the point where the graph of $f(u)$ crosses the u-axis is known. The function value $f(u_0)$ is computed as well as the function derivative value ($f'(u_0)$) at the point u_0. The position where the tangent line $f'(u_0)$ crosses the u-axis is computed from

$$u_1 = u_0 + p \quad \text{where} \quad p = -\frac{f(u_0)}{f'(u_0)} \tag{3.7}$$

The point u_1 serves as a better estimate for the root of the function $f(u)$. The process continues until the root u^* is found. As can be seen from the graph (Figure 3.1) the absolute value of $f(u_1)$ is larger than the absolute value of $f(u_0)$. The algorithm used in this thesis first checks if a better solution is found at the new update. If this is not the case an increment (0.9 times the full update) in the direction of the update from the previous u is tested. The reduction of the full update is repeated until a new update with smaller absolute function value is obtained. This added check provides extra stability to the solution as the root-searching algorithm may diverge if the updated step is too big.

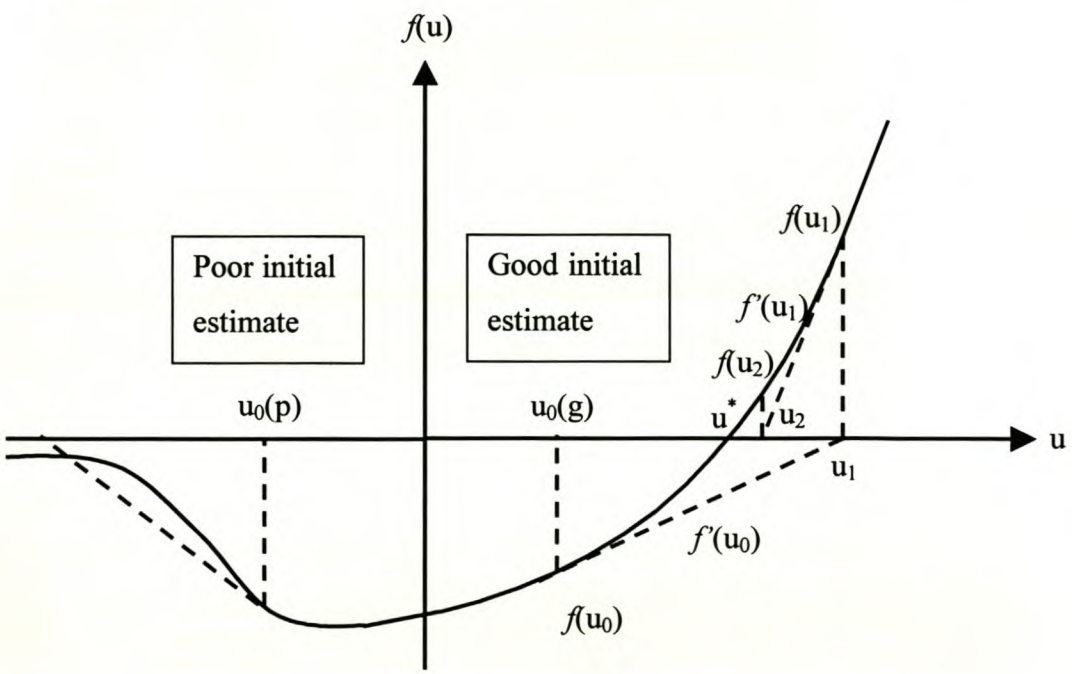

Figure 3.1 Gauss-Newton algorithm

From Figure 3.1 it can also be seen that the Gauss-Newton algorithm will not find the correct solution u^* if poor initial estimates ($u_0(p)$) are made for the parameter values. The Gauss-Newton algorithm will try to find the root solution of $f(u)$ to the left (as seen from the tangent line drawn at $u_0(p)$) of the actual solution u^* in the case where $u_0(p)$ is selected as the initial estimate of the root of $f(u)$.

The Gauss-Newton method is applied to a rectangular system of equations by substituting the Jacobian matrix for the function derivative values and solving a linear least squares system of the form:

$$\mathbf{Jp} = -\mathbf{d} \tag{3.8}$$

where **J** is the m x n Jacobian matrix whose ith row is the gradient of d_i with respect to the parameters u, i.e.:

$$J_{ij} = \frac{\partial d_i}{\partial u_j} \tag{3.9}$$

3.3 Planes

At least three points are required to fit a plane. A plane is specified by:

- a point (x_0, y_0, z_0) on the plane and
- the direction cosines (a, b, c) of the normal to the plane, where $a^2 + b^2 + c^2 = 1$.

Any point (x_i, y_i, z_i) on the plane satisfies:

$$a(x_i - x_0) + b(y_i - y_0) + c(z_i - z_0) = 0 \tag{3.10}$$

The distance from any point (x_i, y_i, z_i) to the plane is found from the dot product:

$$d_i = a(x_i - x_0) + b(y_i - y_0) + c(z_i - z_0) \tag{3.11}$$

The best-fit plane, P, passes through the centroid $(\bar{x},\bar{y},\bar{z})$ of the data and this specifies a point on P. The direction cosines (a,b,c) of P, is the eigenvector associated with the smallest eigenvalue of $\mathbf{B} = \mathbf{A}^{\mathrm{T}}\mathbf{A}$, where $\mathbf{A}$ is the m x 3 matrix with ith row $(x_i - \bar{x}, y_i - \bar{y}, z_i - \bar{z})$ [Forbes, 1991]. The square matrix $\mathbf{B}$ must however be computed to use the eigenvalue solution method. Alternatively, (a,b,c) is the singular vector associated with the smallest singular value of $\mathbf{A}$ when the singular value decomposition method is used. Correct values for the parameters are obtained with a single calculation because this least squares problem is linear. This least squares method was used by Forbes [1991]. A flow chart for the plane fit algorithm is given in Appendix A.

3.4 Spheres

At least four points are required to fit a sphere. A sphere is specified by:

- its centre (x_0,y_0,z_0) and
- its radius r.

Any point on the sphere satisfies:

$$(x-x_0)^2 + (y-y_0)^2 + (z-z_0)^2 = r^2 \tag{3.12}$$

The distance from a point to a sphere is found from

$$d_i = r_i - r \tag{3.13}$$

where

$$r_i = \sqrt{\left[(x_i - x_0)^2 + (y_i - y_0)^2 + (z_i - z_0)^2\right]} \tag{3.14}$$

The sum of all squared distances d_i^2 is minimized in the least squares sense to find the best sphere that fits the data.

$$E = \sum_{i=1}^{m} d_i^2 \tag{3.15}$$

The partial derivatives of d_i (3.13) with respect to the parameters x_o, y_o, z_o and r specify the elements of the Jacobian matrix **J**.

$$\begin{aligned}
\frac{\partial d_i}{\partial x_0} &= -(x_i - x_0)/r_i \\
\frac{\partial d_i}{\partial y_0} &= -(y_i - y_0)/r_i \\
\frac{\partial d_i}{\partial z_0} &= -(z_i - z_0)/r_i \\
\frac{\partial d_i}{\partial r} &= -1
\end{aligned} \tag{3.16}$$

The algorithm to find the best-fit sphere is based on the Gauss-Newton algorithm. Good estimates of the sphere centre (x_0, y_0, z_0) and radius r are necessary to start the iteration. These estimates can be found from a sphere-fitting model solved by a linear least squares problem [Forbes, 1991]. For accurate data, this model should give a best-fit sphere that agrees very closely with the best-fit sphere found by the full non-linear model. The approximate model minimizes

$$F = \sum_{i=1}^{m} f_i^2 \qquad \text{where} \quad f_i = r_i^2 - r^2 \tag{3.17}$$

By a change of parameters, f_i is made a linear function

$$\begin{aligned}
f_i &= (x_i - x_0)^2 + (y_i - y_0)^2 + (z_i - z_0)^2 - r^2 \\
&= -2x_i x_0 - 2y_i y_0 - 2z_i z_0 + (x_0^2 + y_0^2 + z_0^2 - r^2) + (x_i^2 + y_i^2 + z_i^2)
\end{aligned} \tag{3.18}$$

of x_0, y_0, z_0 and $p = x_0^2 + y_0^2 + z_0^2 - r^2$. To minimize F, the linear least squares system

$$\mathbf{A}\begin{bmatrix} x_0 \\ y_0 \\ z_0 \\ p \end{bmatrix} = \mathbf{b} \quad \text{is solved} \tag{3.19}$$

where the elements of the ith row of **A** are the coefficients $(2x_i, 2y_i, 2z_i, -1)$ and the ith element of **b** is $\left(x_i^2 + y_i^2 + z_i^2\right)$. An estimate of r is found from $r = \sqrt{\left(x_0^2 + y_0^2 + z_0^2 - p\right)}$. The approximate model is not used to solve for the sphere parameters since f_i^2 in equation 3.17 will be larger for a point outside the sphere surface than for a point inside the sphere surface when the points are the same distance from the sphere surface. Therefore, the influence of points outside the sphere surface will be larger than that of points inside the sphere surface.

Another method that can be used to find the initial parameters was published by Werghi et al. [1999]. This method uses a general quadratic equation to represent the surface of a sphere:

$$\begin{aligned} f(x,y,z) = ax^2 + by^2 + cz^2 + 2hxy + 2gxz + 2fyz \\ + 2ux + 2vy + 2wz + d = 0 \end{aligned} \tag{3.20}$$

In the case of a sphere the scalar coefficients a, b and c are equal and h, g and f are equal to zero. Therefore

$$f(x,y,z) = a\left(x^2 + y^2 + z^2\right) + 2ux + 2vy + 2wz + d = 0 \tag{3.21}$$

The centre of the sphere is

$$(x_0, y_0, z_0) = \left(-\frac{u}{a}, -\frac{v}{a}, -\frac{w}{a}\right) \tag{3.22}$$

and the radius is

$$r^2 = \frac{u^2 + v^2 + w^2 - ad}{a^2} \tag{3.23}$$

These two methods require that the points used to determine the initial parameters have to be representative of the sphere. The initial parameter values may be computed very inaccurately or even incorrectly if this requirement is not met. Therefore at least four points representative of the sphere have to be selected by the user to solve the initial parameter values. Computing the parameter values with only four points is risky, as a small error in one of the points will strongly influence the parameter values. Therefore it will necessitate the user to select more points (representative of the sphere surface) from the entity surface to compute the initial parameter values correctly. One of the thesis aims was to reduce the reverse engineering time by reducing user input. This lead to the use of a geometrical method.

Two points are selected by the user and the neighbours of these points are found. A plane is then constructed at both selected points on the sphere surface. Figure 3.2 shows the two points selected and their respective constructed planes *A* and *B*. The normal direction vectors of both planes are found and placed at the centroid of each plane. Lagrange optimisation [Rao, 1996] is used to find the shortest line between these to normal direction vectors. The point where the unit normal intersects plane *A* is called (x_{01}, y_{01}, z_{01}). The same is done for plane *B* to obtain the point (x_{02}, y_{02}, z_{02}). A point (x_1, y_1, z_1) exists on the direction normal vector of plane *A* a distance λ_1 from point x_{01}, y_{01}, z_{01}. Another point (x_2, y_2, z_2) exists on the direction normal vector of plane *B* a distance λ_2 from point (x_{02}, y_{02}, z_{02}).

Therefore the following equations hold:

$$\begin{pmatrix} x_1 \\ y_1 \\ z_1 \end{pmatrix} = \begin{pmatrix} x_{01} \\ y_{01} \\ z_{01} \end{pmatrix} + \lambda_1 \cdot \begin{pmatrix} a_1 \\ b_1 \\ c_1 \end{pmatrix} \tag{3.24}$$

and
$$\begin{pmatrix} x_2 \\ y_2 \\ z_2 \end{pmatrix} = \begin{pmatrix} x_{02} \\ y_{02} \\ z_{02} \end{pmatrix} + \lambda_2 . \begin{pmatrix} a_2 \\ b_2 \\ c_2 \end{pmatrix} \tag{3.25}$$

where (a_1, b_1, c_1) are the direction cosines of the normal of plane *A* and (a_2, b_2, c_2) are the direction cosines of the normal of plane *B*. The minimization function is the distance between point (x_1, y_1, z_1) and point (x_2, y_2, z_2).

$$f = (x_2 - x_1)^2 + (y_2 - y_1)^2 + (z_2 - z_1)^2 \tag{3.26}$$

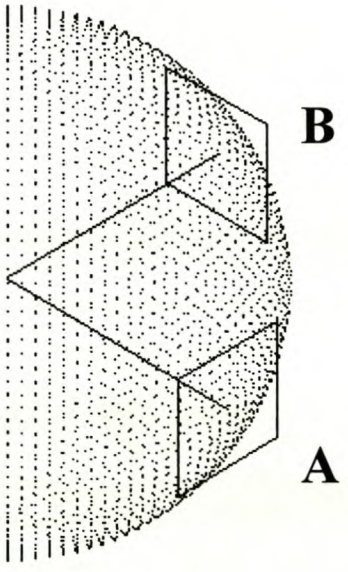

Figure 3.2 Estimation of sphere parameters

The derivative of *f* with respect to λ_1 and λ_2 is calculated and the result equated to zero. By substitution, equations for λ_1 and λ_2 are found. The shortest line is given by its two end points (x_1, y_1, z_1) and (x_2, y_2, z_2). This line is always normal to both the normal direction vectors of plane *A* and plane *B*. The sphere centre is approximated by the midpoint of the shortest line. Note that the centre of a full sphere with equally distributed points, is the centroid of the points on the sphere surface. The radius is found from the average absolute distances of all the data points (that were used to estimate the initial sphere parameters) to the estimated sphere centre. The sphere

parameters cannot be approximated accurately if the two start points selected lie too close to each other or on opposite sides of the sphere. The algorithm checks the dot product between the two plane normal vectors. It warns the user if the absolute value of the dot product is bigger or equal to 0.95.

One iteration of the Gauss-Newton algorithm [Forbes, 1991] to solve the non-linear model is given below:

- solve the linear least squares system

$$\mathbf{J}\begin{bmatrix} p_{x_0} \\ p_{y_0} \\ p_{z_0} \\ p_r \end{bmatrix} = -\mathbf{d} \quad \text{where } \mathbf{J} \text{ and } \mathbf{d} \text{ are given by (3.16) and (3.13).} \tag{3.27}$$

- update the parameter estimates according to

$$\begin{aligned} x_o^{k+1} &= x_o^k + p_{x_0} \\ y_o^{k+1} &= y_o^k + p_{y_0} \\ z_o^{k+1} &= z_o^k + p_{z_0} \\ r^{k+1} &= r^k + p_r \end{aligned} \tag{3.28}$$

These steps are repeated until the algorithm has converged. The algorithm has converged when the size of the update, measured by $\sqrt{\mathbf{p}^T\mathbf{p}}$ is small. A flow chart for the sphere fit algorithm is given in Appendix A.

3.5 Cylinders

At least five points are required to fit a cylinder. A cylinder is specified by:

- a point (x_0, y_0, z_0) on its axis,
- a vector (a, b, c) pointing along the axis and
- its radius r.

To parameterise a cylinder properly a systematic way is needed to decide which point on the axis to choose, along with a constraint on (a,b,c). Vertical cylinders are characterized by

$$z_0 = -ax_0 - by_0 \text{ and } c = 1. \tag{3.29}$$

The distance from a point to a cylinder is found from

$$d_i = r_i - r \tag{3.30}$$

where

$$r_i = \frac{\sqrt{(u_i^2 + v_i^2 + w_i^2)}}{\sqrt{(a^2 + b^2 + c^2)}} \tag{3.31}$$

with

$$\begin{aligned} u_i &= c(y_i - y_0) - b(z_i - z_0) \\ v_i &= a(z_i - z_0) - c(x_i - x_0) \\ w_i &= b(x_i - x_0) - a(y_i - y_0) \end{aligned} \tag{3.32}$$

The sum of all squared distances d_i^2 is minimized in the least squares sense to find the best cylinder that fits the data.

$$E = \sum_{i=1}^{m} d_i^2 \tag{3.33}$$

r_i is the distance of the ith point to the cylinder axis. Employing the constraints (3.29), d_i is a function of the five parameters x_0, y_0, a, b and r, and to implement a Gauss-Newton algorithm to minimize the sum of squares of the distances, the partial derivatives of d_i with respect to these five parameters must be calculated. The

expressions are complex but simplifies considerably for the special case when $x_0 = y_0 = a = b = 0$. The distance from any point to the cylinder axis is then given by

$$r_i = \sqrt{\left(x_i^2 + y_i^2\right)} \tag{3.34}$$

and the *i*th row of the Jacobian matrix **J** is equal to

$$\begin{aligned}
\frac{\partial d_i}{\partial x_0} &= -x_i / r_i \\
\frac{\partial d_i}{\partial y_0} &= -y_i / r_i \\
\frac{\partial d_i}{\partial a} &= -x_i z_i / r_i \\
\frac{\partial d_i}{\partial b} &= -y_i z_i / r_i \\
\frac{\partial d_i}{\partial r} &= -1
\end{aligned} \tag{3.35}$$

The algorithm description follows. Good initial estimates of a point on the axis (x_0,y_0,z_0), the axis vector (a,b,c) and the cylinder radius r are needed. Methods to find these estimates will be discussed later in this section.

One iteration of the Gauss-Newton algorithm to solve the non-linear model is given below:

- translate the data so that the point on the axis lies at the origin:

$$\left(x'_i, y'_i, z'_i\right) = \left(x_i, y_i, z_i\right) - \left(x_0, y_0, z_0\right) \tag{3.36}$$

- transform the data by a rotation matrix **U** [Forbes, 1991] which rotates (a,b,c) to a point on the z-axis:

$$\begin{bmatrix} x_i'' \\ y_i'' \\ z_i'' \end{bmatrix} = \mathbf{U} \begin{bmatrix} x_i' \\ y_i' \\ z_i' \end{bmatrix} \qquad \text{where} \tag{3.37}$$

$$\mathbf{U} = \begin{bmatrix} c_2 & 0 & s_2 \\ 0 & 1 & 0 \\ -s_2 & 0 & c_2 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & c_1 & s_1 \\ 0 & -s_1 & c_1 \end{bmatrix} \quad \text{with} \tag{3.38}$$

$$c_1 = \cos(\theta_1),\ s_1 = \sin(\theta_1),\ c_2 = \cos(\theta_2),\ s_2 = \sin(\theta_2) \text{ and} \tag{3.39}$$

$$\theta_1 = \arctan\left(\frac{-b}{c}\right) \tag{3.40}$$

$$\theta_2 = \arctan\left(\frac{a}{b\sin(\theta_1) - c\cos(\theta_1)}\right) \tag{3.41}$$

- solve the linear least squares system:

$$\mathbf{J} \begin{bmatrix} p_{x_0} \\ p_{y_0} \\ p_a \\ p_b \\ p_r \end{bmatrix} = -\mathbf{d} \quad \text{where } \mathbf{J} \text{ and } \mathbf{d} \text{ are given by (3.35) and (3.30).} \tag{3.42}$$

- update the parameter estimates according to

$$\begin{bmatrix} x_0^{k+1} \\ y_0^{k+1} \\ z_0^{k+1} \end{bmatrix} = \begin{bmatrix} x_0^{k} \\ y_0^{k} \\ z_0^{k} \end{bmatrix} + \mathbf{U}^T \begin{bmatrix} p_{x_0} \\ p_{y_0} \\ -p_{x_0} p_a - p_{y_0} p_b \end{bmatrix} \tag{3.43}$$

$$\begin{bmatrix} a \\ b \\ c \end{bmatrix} = \mathbf{U}^T \begin{bmatrix} p_a \\ p_b \\ 1 \end{bmatrix} \text{ and} \tag{3.44}$$

$$r = r + p_r \tag{3.45}$$

These steps are repeated until the algorithm has converged. The algorithm has converged when the size of the update, measured by $\sqrt{\mathbf{p}^T\mathbf{p}}$ is small. A flow chart for the cylinder fit algorithm is given in Appendix A.

The point (x_0, y_0, z_0) can be translated to a point (still on the axis of the cylinder) near to the origin of the coordinate system by the following transformation after the final entity parameter values are found:

$$\begin{bmatrix} x_{0'} \\ y_{0'} \\ z_{0'} \end{bmatrix} = \begin{bmatrix} x_0 \\ y_0 \\ z_0 \end{bmatrix} - \frac{ax_0 + by_0 + cz_0}{a^2 + b^2 + c^2} \begin{bmatrix} a \\ b \\ c \end{bmatrix} \tag{3.46}$$

The procedure to find good initial estimates of a point on the axis (x_0, y_0, z_0), the axis vector (a, b, c) and the cylinder radius r follows.

A general quadric surface is represented by the following quadratic equation:

$$\begin{aligned} f(x, y, z) = ax^2 + by^2 + cz^2 + 2hxy + 2gxz + 2fyz \\ + 2ux + 2vy + 2wz + d = 0 \end{aligned} \tag{3.47}$$

which can be written as

$$\mathbf{x}^T\mathbf{A}\mathbf{x} + 2\mathbf{x}^T\mathbf{b} + C = 0 \tag{3.48}$$

where

$$\mathbf{A} = \begin{bmatrix} a & h & g \\ h & b & f \\ g & f & c \end{bmatrix} \qquad \mathbf{b} = \begin{bmatrix} u \\ v \\ w \end{bmatrix} \qquad C = d \qquad \mathbf{x} = \begin{bmatrix} x \\ y \\ z \end{bmatrix} \tag{3.49}$$

The type of quadric depends on the determinant of the quadric Δ, the cubic determinant D and the cofactors of D.

$$\Delta = \begin{vmatrix} a & h & g & u \\ h & b & f & v \\ g & f & c & w \\ u & v & w & d \end{vmatrix} \qquad D = \begin{vmatrix} a & h & g \\ h & b & f \\ g & f & c \end{vmatrix} \tag{3.50}$$

$$\begin{aligned} d_1 &= bc - f^2 & d_2 &= ac - g^2 & d_3 &= ab - h^2 \\ d_4 &= gh - af & d_5 &= hf - bg & d_6 &= gf - ch \end{aligned} \tag{3.51}$$

The best parameters that satisfy for n data points in the least squares sense [Werghi et al., 1999] are those minimizing the criterion

$$\sum_{i=1}^{n} f(x_i, y_i, z_i)^2 = \mathbf{p}^T \left(\sum_{i=1}^{n} \mathbf{h}_i \mathbf{h}_i^T \right) \mathbf{p} = \mathbf{p}^T \mathbf{H} \mathbf{p} \tag{3.52}$$

$$\text{where } \mathbf{p} = \begin{bmatrix} a \\ b \\ c \\ h \\ g \\ f \\ u \\ v \\ w \\ d \end{bmatrix} \qquad \mathbf{h}_i = \begin{bmatrix} x_i^2 \\ y_i^2 \\ z_i^2 \\ 2x_i y_i \\ 2x_i z_i \\ 2y_i z_i \\ 2x_i \\ 2y_i \\ 2z_i \\ 1 \end{bmatrix} \tag{3.53}$$

The quadric is a cylinder when Δ = D = 0, $ud_1 + vd_6 + wd_5 = 0$, $d_1 + d_2 + d_3 > 0$, $agh + f(g^2 + h^2) = 0$, $cfg + h(f^2 + g^2) = 0$, $bhf + g(f^2 + h^2) = 0$ and $\frac{u}{f} + \frac{v}{g} + \frac{w}{h} = 0$. The equation of the cylinder axis is:

$$\frac{x-\frac{uf}{d_4}}{1/d_4}=\frac{y-\frac{vg}{d_5}}{1/d_5}=\frac{z-\frac{wh}{d_6}}{1/d_6} \tag{3.54}$$

This means that the cylinder axis has the direction vector $[1/d_4, 1/d_5, 1/d_6]^T$ and passes through the point $[(uf/d_4),(vg/d_5),(wh/d_6)]^T$. The axis orientation corresponds to the eigenvector related to the null eigenvalue of the matrix **A**. The other two eigenvalues λ are equal and positive. The radius can be expressed by

$$r^2=\left(u^2 f/d_4+v^2 g/d_5+w^2 h/d_6+d\right)/\lambda \tag{3.55}$$

This method described by Werghi et al. [1999] requires ten points to determine the ten quadric parameters of equation 3.47. Another method similar to the one used to find the initial sphere parameters, is used in this thesis to find the initial cylinder parameters. Two points are selected by the user and the neighbours of these points are found. A plane is then constructed at each of the selected points on the cylinder surface as shown if Figure 3.3.

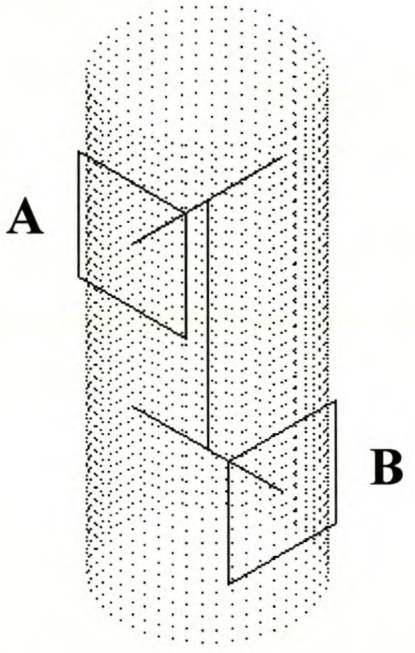

Figure 3.3 Estimation of cylinder parameters

The normal direction vectors of both planes are found and placed at the centroid of each plane. The shortest line between these two normal vectors is computed as previously explained. The two computed normal vectors will go through the cylinder axis and will be normal to the axis if the data consist of exact points (a perfect cylinder with no measuring errors in the data) and the point distribution is exactly uniform. The shortest line is also perpendicular to the two normal vectors and therefore it is a good approximation of the cylinder axis direction. The midpoint of this shortest line is assigned as a point on the axis (x_0, y_0, z_0). The cylinder radius is found from the average absolute distances (Equation 3.31) of all the data points (that were used to estimate the initial cylinder parameters) to the estimated cylinder axis.

Poor approximations of the cylinder parameters may be obtained if the two points used lie too close to each other. The two normal vectors of the constructed planes will be almost parallel if the points (used to construct the planes) lie on the same generator line or on different generator lines that are directly across each other. The user is warned if the absolute value of the dot product of the two plane normal vectors is bigger or equal to 0.95. Another scenario to be avoided is that the two selected points lie in the same cross section of the cylinder. This will influence the accuracy to which the cylinder axis direction will be calculated. The user are warned if these two points lie closer than twice the average point pitch in the cylinder axis direction from each other. Good parameter estimates will be obtained if these two rules are satisfied.

4 Data Structure and Neighbour Search Routines

This chapter describes the octree data structure and how it is used. Once the octree nodes are initialised, it is necessary to know which nodes lie next to a specific node. These nodes are called neighbours. Search routines to find neighbour nodes are also discussed in this chapter.

4.1 The Octree Data Structure

The octree data structure (adapted from Jones [1999]) provides a very efficient way to search for specific points among large point data sets. Point cloud is used as a synonym for large point data sets throughout the thesis. This data structure eliminates the work of searching through the whole data set each time when a specific point has to be found. Therefore this chosen structure saves on computer time.

The algorithm first reads the point data from a text file and stores it in a data list. The size of a box that will contain all the data is then determined. It searches for the largest and smallest x, y and z value of all the data points in the point cloud. The six values found are used to determine a box around the point cloud. The box is made slightly bigger than the box determined by the coordinate values. This is to ensure that no points lie in the six side planes of the box. The distance added to each side of the original determined box is equal to one hundredth of the average pitch (distance between the points in mm) of the point data. The search algorithm (described in Chapter 5) needs cubically refined boxes. The reason for this is that the entity search algorithm has to search at an equal rate in all directions to find an entity. The node dimensions are used as a search step and therefore the initially determined node has to be a cube. Therefore the originally determined box (not necessarily a cube) is expanded so that all sides have a length equal to the longest side of the original box. This cube is called the root node.

The root node (parent) is divided into eight smaller nodes (children) as shown in Figure 4.1. It is first divided along the x-axis, then the y-axis and lastly along the z-axis. Each of the children is then also divided into eight nodes. This dividing process

continues until refinement criteria are met. Each child node will have one parent node and each parent node will have eight smaller child nodes that may be parent nodes for other children. An octree-node can be either an internal node or a leaf node. Internal nodes contain no points but have pointers to their child nodes. Leaf nodes have no child nodes but have a pointer to the first data point that belongs to the node in an array of data points.

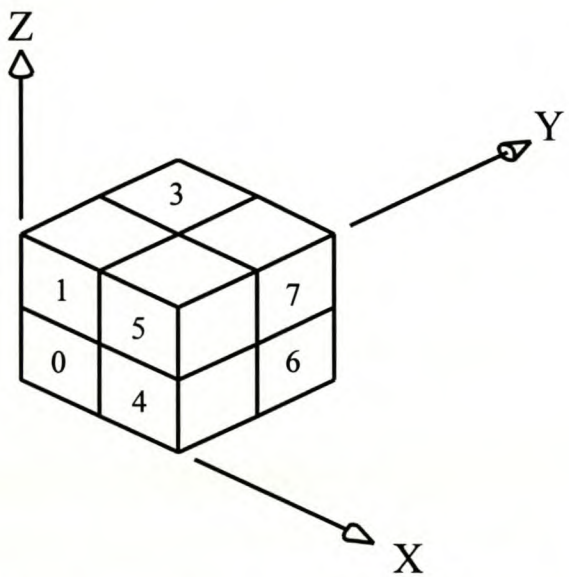

Figure 4.1 Octree node division and node numbering

Janssens [1998] suggested that the refinement should continue until the length of the shortest side of the child node is three times the average pitch of the data or until the node contains only one point. This second part of the stop criteria is altered in this thesis so that refinement continues until a node contains no points. This is necessary because the search algorithm uses the node size as a search distance for points belonging to an entity.

Consider the two objects shown in Figure 4.2. One is part of a cylinder and the other part of a cube at an angle of 45°. The two objects end in the XY plane. Therefore their ends lie on the same height in the Z direction. A search for the flat circular end plane of the cylinder will include nodes A1, A2, A3 and A4. If the stop criteria allow the algorithm to stop refining nodes if it contains only one point, node B3 is refined as shown in Figure 4.2. The algorithm will extend it search from here and will include node B3 and in the next iteration nodes B1, B2 and B4 because the node size is used

as a search step. This is why the stop criteria are altered so that the nodes are refined until it contains no points. Therefore node B3 will be refined once more and node A2 will have an empty node to its top right hand corner. This prevents the algorithm from including the wrong nodes while searching for the entity.

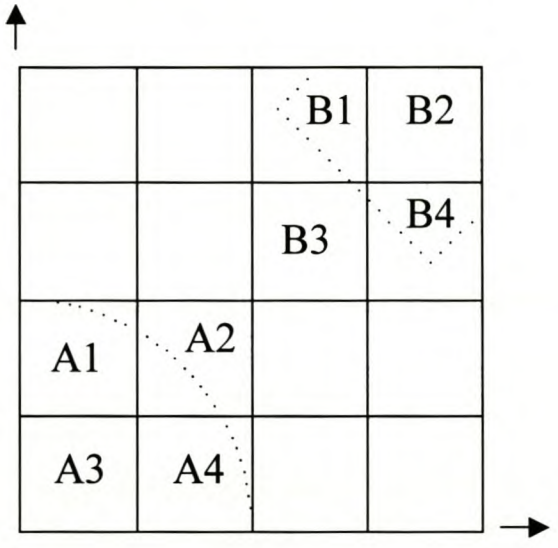

Figure 4.2 Octree nodes with two different entities in the same plane

The octree division process is explained with the aid of Figure 4.3. It shows the root node and divisions up to the third level. Nodes that do not contain points are not shown for clarity. The octree structure of Figure 4.3 is shown schematically in Figure 4.4. Internal nodes are black throughout and leaf nodes are shown as dots with outlines only. Leaf nodes with points are grey. Child nodes zero and seven of the root node are refined to level one. At level two child node seven of each of the previous two nodes are refined. This leads to leaf nodes six and one containing points at level three.

The octree data structure algorithm is explained in detail in Appendix A. A flow chart is also presented.

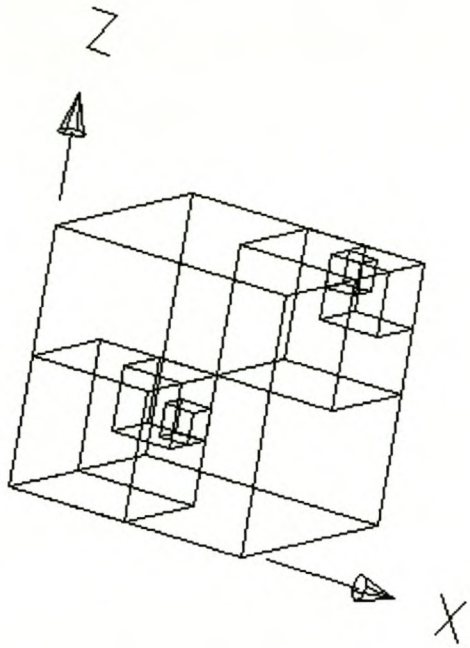

Figure 4.3 Octree root node with child nodes

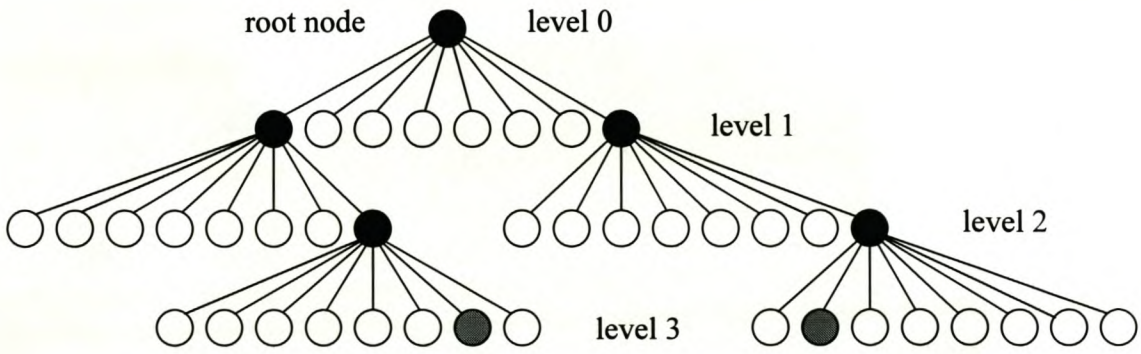

Figure 4.4 Schematical representation of octree data structure of Figure 4.3

4.2 Neighbour Search Routines

A specific point can easily be found with the octree algorithm by searching from the root node down in the tree until the leaf node containing that point is found. All other points in that child node can then easily be found from the data array. A problem arises when points in a node adjacent to this leaf node have to be found. No node neighbour information is conveyed by the octree data structure. This problem is

solved by storing binary position numbers for each node and employing a node neighbour search routine. Each neighbour node has a specific location name and is placed in one of three categories.

The following neighbours exist for a node:

- Neighbour nodes that only differ in one axis direction as shown in Figure 4.5.
- Neighbour nodes that differ in two axis directions as shown in Figure 4.6.
- Neighbour nodes that differ in three axis directions as shown in Figure 4.7.

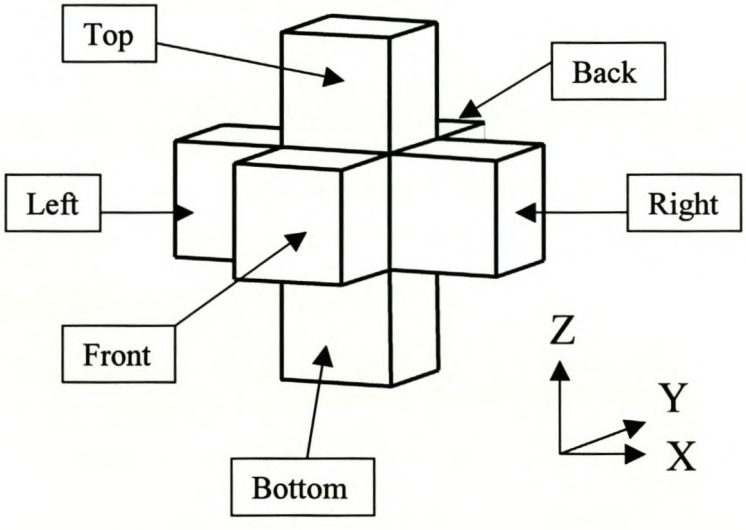

Figure 4.5 Neighbour nodes that only differ in one axis direction

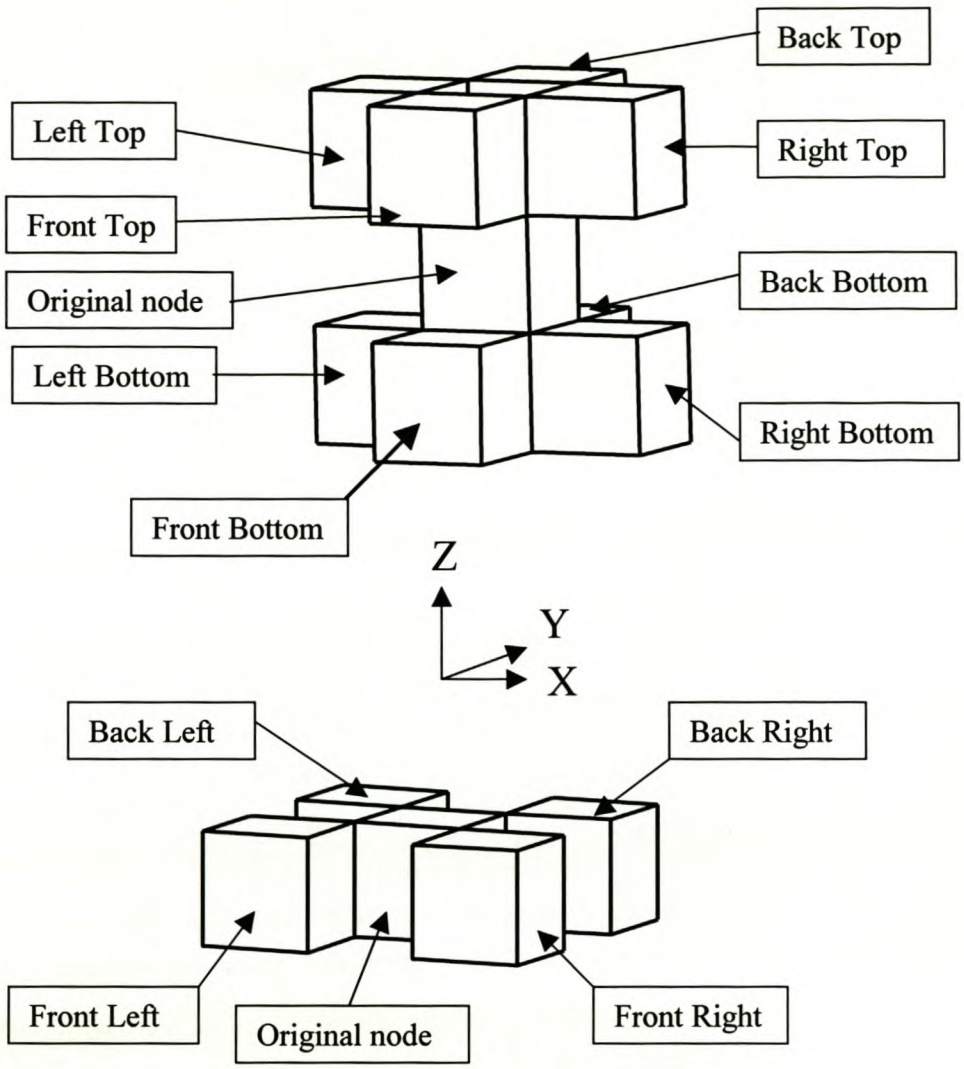

Figure 4.6 Neighbour nodes that differ in two axis directions

Vörös [2000] described a method for finding neighbours in an octree data structure. Vörös used six search directions to find neighbours of a specific node. These six direction neighbours correspond to the neighbours shown in Figure 4.5. This method is altered so that all twenty-six neighbours (Figures 4.5 – 4.7) can be found directly when all the neighbours are of equal size.

Binary position numbers are assigned to all nodes when the octree is created. Table 4.1 shows the binary numbers for the nodes displayed in Figure 4.1 for the first division of the octree root node.

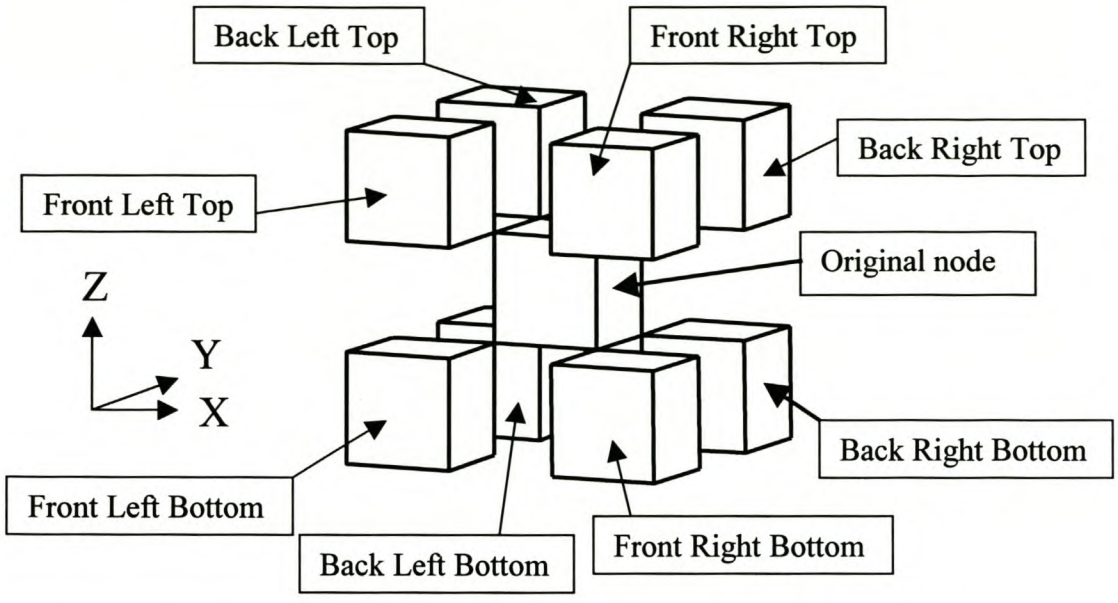

Figure 4.7 Neighbour nodes that differ in three axis directions

Node	x-position	y-position	z-position
0	0	0	0
1	0	0	1
2	0	1	0
3	0	1	1
4	1	0	0
5	1	0	1
6	1	1	0
7	1	1	1

Table 4.1 Binary node numbers for specific node positions

As the octree is refined, more binary digits are added to the corresponding binary position number for a specific node. The root node of the octree is at division level zero. The first division creates nodes at division level one and so forth. A node refined to division level four, would have four digits representing each of the three coordinate positions.

The left leaf node of Figure 4.3 has a creation number of 076. The creation number has the same number of digits as the level number of the node. A node divided up to the third level will therefore have three digits in its creation number. The digit values will range from 0 to 7 and this relate to the position of the node at that division level. This means that to find the leaf node a search must be done starting at the root node, finding its first child node (0), finding the last child node (7) of child node 0 and finding the second last child node (6) of child node 7. According to the scheme of Table 1 this leaf node will have a binary x-position number of 011, a binary y-position number of 011 and a binary z-position number of 010. The right leaf node of Figure 4.3 will have a binary x-position number of 110, a binary y-position number of 110 and a binary z-position number of 111.

A node can have either an inner neighbour (neighbour belongs to the same parent) or an outer neighbour (neighbour belongs to a different parent) in a certain search direction. One binary digit is added to the x, y or z node position number if a neighbour lies in the positive x-, y- or z-direction. One binary digit is subtracted for the negative directions. Inner neighbours are easily found by changing the last digit of the corresponding position number. Outer neighbours are found by searching for the common parent of both nodes. The algorithm proceeds by searching, from the common parent, down the octree (increasing levels) until no more neighbour nodes in the corresponding direction can be found. This method can be used to search for smaller, equal or larger neighbour nodes.

Equal size neighbours are found easily as their level in the octree are equal. Once a common parent is found the algorithm searches down the octree until a neighbour node is found with the same level as the original node. Larger nodes are identified as the algorithm first checks if a node has children before it searches down the octree. If a larger neighbour exists for a certain node, the algorithm will stop at that node as it contains no children. Smaller neighbour nodes are found by checking if a neighbour node at the same level as the original node has any children. Only the smaller neighbour children are found if the equal sized neighbour has children. A recursive algorithm is applied to search for smaller sized neighbours. The neighbour search routine is described in more detail with the aid of a flow chart in Appendix A.

Another neighbour scheme algorithm was implemented that uses the node creation numbers as shown in Figure 4.1. This method is clumsy as the algorithm cannot add or subtract integers from the node creation number to find neighbours in the positive and negative directions.

The following example finds the outer corner (top right) neighbour of *A* that is a smaller node than *A* in Figure 4.8. This example is illustrated for a two-dimensional case for clarity, therefore the z position values can be ignored. Figure 4.8 represents the top middle part of the root node refined up to three levels. The thickest lines (cross) represent the first division and the mid-thickness lines (all around the sides) represent the second division. The third division is shown by the finest lines.

The algorithm adds 1 to the x- and y-position binary number of *A* resulting in a neighbour node with x-position binary number equal to 10 and y-position binary number equal to 10. The first binary numbers of both positions have changed. This means that the root node is the parent node to which the selected node and its neighbour belong. The algorithm searches up the tree until it reaches the root node and continues by searching down the tree until it finds the node corresponding to the neighbour binary number (top right node 1 in Figure 4.8). It checks if this found neighbour node has any children. This neighbour has children and therefore it searches for the node nearest to the selected node. In this case, it is the node with x-position equal to 100 and y-position equal to 100. This node has no children and therefore the algorithm stops.

The same figure is used to explain the neighbour search algorithm if node creation numbers would have been used. Node *A* has a node creation number of 1-7. Its top right neighbour has a node creation number of 7-1-1. This neighbour is found by searching for the top right node of a node with creation number 1 at level 1 with creation number of 7. The next step is to realise that a node with a creation number of 7 in the second division level must have a top right neighbour node with a creation number of 1. As this neighbour node has children, the touching neighbour is once more the node with a creation number of 1 at the third division level. This node is the neighbour node searched for. A large number of computations have to be done to find

all twenty-six neighbours of a specific node in all directions when this scheme is followed. This is why the scheme using the binary creation numbers was chosen.

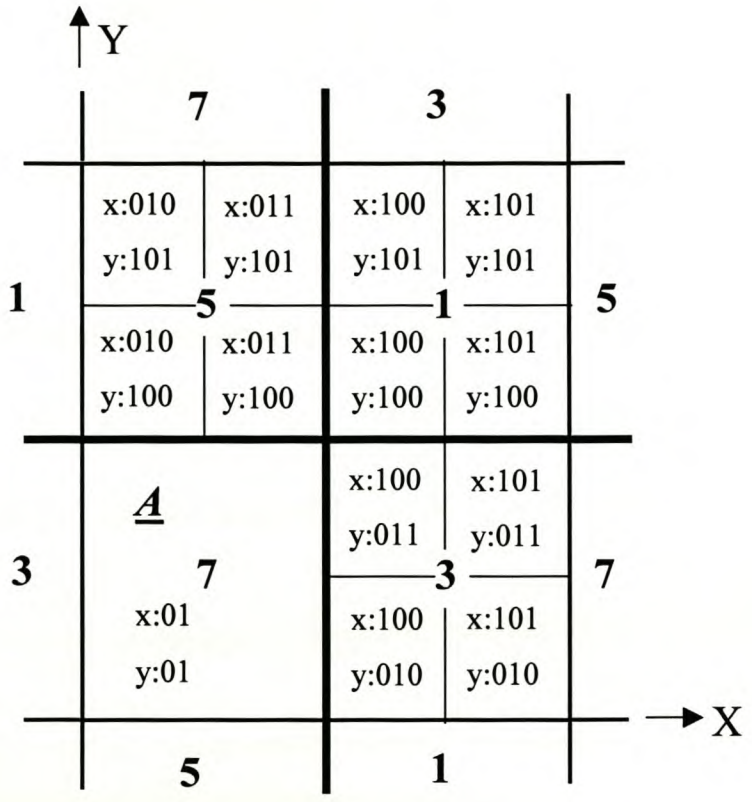

Figure 4.8 Octree nodes belonging to different parents

A three-dimensional example is also presented for completeness. Refer to Figure 4.9. The lighter grey back left top neighbour node of the original darker grey node will be found. The lighter grey node has a node creation number of 0-7-4 and binary position numbers x:011, y:010 and z:010. The darker grey node has a node creation number of 4-0-3 and binary position numbers x:100, y:001 and z:001. The lighter grey neighbour lies to the left in the x-direction of the original darker node. This means that one binary digit must be subtracted from the x-position binary number of the darker grey node. This gives x:011. One binary digit must be added to the y-position binary number as the lighter grey node lies in the positive y-direction. This gives y:010. Lastly, one binary digit must be added to the z-position binary number for the same reason giving z:010. Table 4.2 summarise the results.

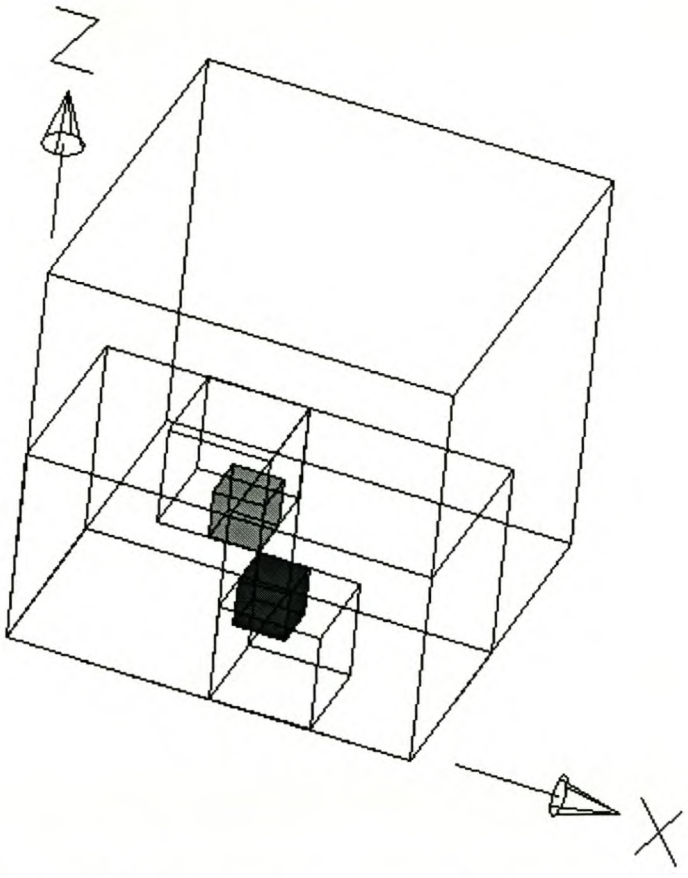

Figure 4.9 Neighbour nodes at creation level of three

		Darker grey original node	Lighter grey neighbour
x-position	binary	100	011
y-position	binary	001	010
z-position	binary	001	010

Table 4.2 Summary of node binary numbers for three-dimensional example

The first column of the darker grey original node is 1-0-0 and the first column of the lighter grey neighbour node is 0-0-0. this means that the common parent node of both nodes is the root node. The algorithm searches up the octree until the root node is found and the down the octree for the node with binary position numbers of x:011, y:010 and z:010. This gives the lighter grey back left top neighbour node. The common parent is always specified by the last column where the binary numbers are the same.

5 Entity Search Algorithm

The entity search algorithm enables the user to select specific points from the point cloud and grow an entity (planes, spheres or cylinders, as specified by the user) from those selected points. The algorithm tries to find the best entity that will fit the data points and to predict the boundaries of the entity accurately. User input influences the grow process significantly and therefore Chapter 6 provides information on how to choose good values for the user input that will ensure that the best entity is fitted. An accurate entity is measured on how well it fits the original entity and how well the entity boundary is found. The algorithm aims are to be robust in the searching process and to be as accurate as possible.

5.1 Overall Strategy

The user starts the entity extraction program and selects a text file containing the point cloud. The algorithm creates an octree data structure to manage the point data. The data points are displayed on the screen and the user must then select one or more points, depending on the type of entity that has to be fitted. Using the selected points and other points in their vicinity, initial estimates of the entity parameter values are calculated. At this stage a systematic entity grow algorithm can commence.

(a) Neighbour nodes of octree nodes that are already included are identified as candidates for inclusion.
(b) The points in the candidate nodes are tested (based on the most recently fitted geometric entity) to see if they can be included in the entity.
(c) Adding nodes continues (using the previous two steps) until no more nodes are found that can be added to the currently fitted entity.
(d) New entity parameters are now obtained by refitting the entity to the currently included points.
(e) The procedure returns now to evaluating nodes that are candidates for inclusion, step (a).
(f) The grow algorithm stops when no more nodes can be found that belong to the entity.

(g) The final entity fit parameter values are calculated.

(h) The boundary of the entity is found.

5.2 Recursive Grow Algorithm

This paragraph explains, in detail, the different actions necessary to extract the entity as mentioned briefly in the previous paragraph. Reasons for the different steps mentioned are given. User selected input parameters as well as the influence of these parameters on the grow algorithm are described.

5.2.1 Construction of Octree Data Structure

The octree data structure is created with the average pitch size and node size provided by the user. The average pitch size (distance between the points in mm) is determined by the point scanning machine during the data acquisition stage. The length or radius of the entity must be about ten times the size of the average point pitch to ensure the robustness of the grow algorithm.

The node size divided by the average point pitch is normally taken as three as proposed by Janssens [1998]. Octree nodes will be refined until one of the node sides are equal or smaller than this value times the average point pitch or until a node contain no points. This integer number also specifies the size of the search step. If the node size is too big, too many points will be added to the entity during a certain iteration step. This will influence the accuracy to which the geometric entity boundary can be computed as well as the fit accuracy of the final entity. Computer time to build the octree will increase if the node size is made smaller. The node size must be bigger than the average point pitch, otherwise nodes may exist that do not contain points but still belong to a specific entity. This will cause the algorithm to stop searching before the entire entity is found.

5.2.2 Selection of Start Points and Calculation of Initial Parameter Values

The point cloud is displayed in AutoCAD by using the AutoCAD interface described in Appendix A. The user has to select good start points, from the point cloud on the

screen, that do not lie too close to a possible boundary of the entity. If start points are selected too close to the entity boundary, the possibility exists that points that do not belong to a cylinder or a sphere will be included to fit the two planes [Figures 3.1 and 3.2] that determines the initial sphere or cylinder parameters. This start points requirement will improve the chances of fitting the entity correctly. The selected start points are then used by the entity extraction program to compute initial values for the entity parameters as described in Chapter 3.

The user has to supply one start point more or less in the middle of a plane to fit a plane. Two or three points can be supplied to fit a cylinder or a sphere. The first two points are used to find the entity parameter values and if a third point is provided this point will be used as a start point for the grow algorithm. A start point for the entity will be calculated if a third point is not specified. Figures 5.1 and 5.2 show how this third point is calculated for the case of a cylinder and a sphere respectively.

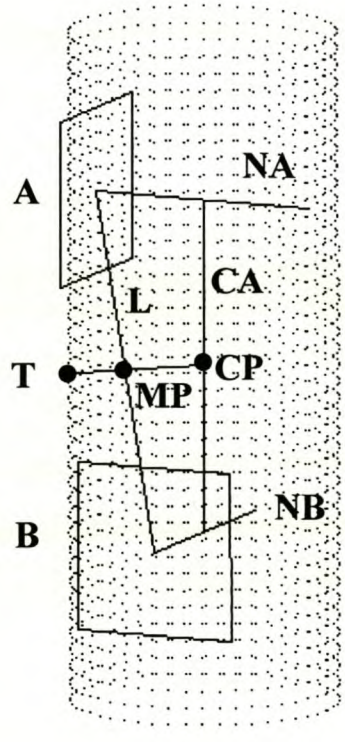

Figure 5.1 Finding the cylinder start point for the grow algorithm if it is not supplied by the user

The first two points selected is used to construct the two planes *A* and *B* as explained in Chapter 3. Both plane normal vectors are constructed through their respective centroids. From this the axis of the cylinder is predicted. Therefore, the direction (a,b,c) as well as a point on the axis (x_0, y_0, z_0) is known. This algorithm constructs a line between the two plane centroids. The midpoint of this line is labelled *MP*. Label the centroid of plane *A* as (x_1, y_1, z_1) and the centroid of plane *B* as (x_2, y_2, z_2). Therefore, *MP* will have the following coordinates:

$$\begin{aligned} x_3 &= (x_1 + x_2)/2 \\ y_3 &= (y_1 + y_2)/2 \\ z_3 &= (z_1 + z_2)/2 \end{aligned} \tag{5.1}$$

The distance *d* from this point to the point on the cylinder axis in the direction of the cylinder axis is computed. This gives:

$$d = a(x_3 - x_0) + b(y_3 - y_0) + c(z_3 - z_0) \tag{5.2}$$

The point *CP* (x_4, y_4, z_4) is now computed on the cylinder axis, a distance *d* from the point (x_0, y_0, z_0).

$$\begin{pmatrix} x_4 \\ y_4 \\ z_4 \end{pmatrix} = \begin{pmatrix} x_0 \\ y_0 \\ z_0 \end{pmatrix} + d \begin{pmatrix} a \\ b \\ c \end{pmatrix} \tag{5.3}$$

A line between *CP* and *MP* is perpendicular to the cylinder axis. The direction vector (a_{34}, b_{34}, c_{34}) between these two points is computed and the result is:

$$\begin{pmatrix} a_{34} \\ b_{34} \\ c_{34} \end{pmatrix} = \begin{pmatrix} x_3 \\ y_3 \\ z_3 \end{pmatrix} - \begin{pmatrix} x_4 \\ y_4 \\ z_4 \end{pmatrix} \tag{5.4}$$

This direction vector is normalized so that

$$a_{34}^2 + b_{34}^2 + c_{34}^2 = 1 \tag{5.5}$$

The start point $T\ (x_5, y_5, z_5)$ is computed by finding the point in the direction of (a_{34}, b_{34}, c_{34}) that is a distance equal to the cylinder radius from the point *CP*.

$$\begin{pmatrix} x_5 \\ y_5 \\ z_5 \end{pmatrix} = \begin{pmatrix} x_4 \\ y_4 \\ z_4 \end{pmatrix} + r \begin{pmatrix} a_{34} \\ b_{34} \\ c_{34} \end{pmatrix} \tag{5.6}$$

The same strategy is employed to find the start point if it is not supplied by the user in the case of a sphere. Figure 5.2 shows a sphere with the two planes constructed to find the initial sphere parameters as explained in Chapter 3.

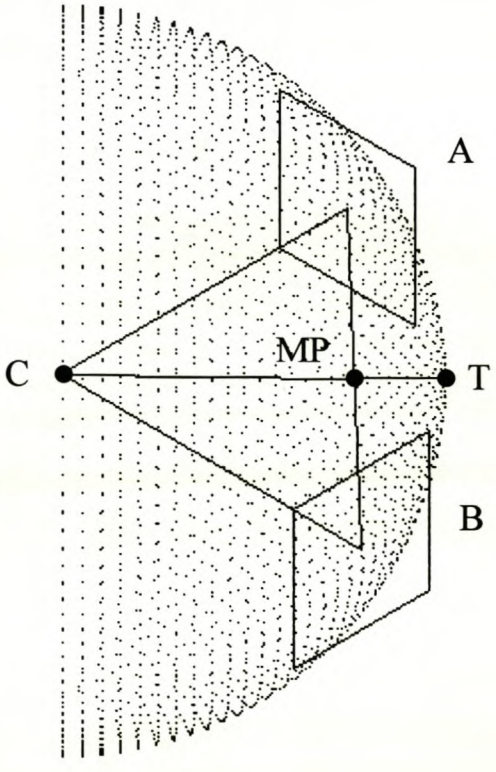

Figure 5.2 Finding the sphere start point for the grow algorithm if it is not supplied by the user

The first two points selected is used to construct the two planes A and B as explained in Chapter 3. Both plane normal vectors are constructed through their respective centroids. From this the sphere centre (x_0, y_0, z_0) and radius is predicted. This algorithm constructs a line between the two plane centroids. The midpoint of this line is labelled *MP*. Label the centroid of plane *A* as (x_1, y_1, z_1) and the centroid of plane *B* as (x_2, y_2, z_2). Therefore, *MP* will have the coordinates:

$$\begin{aligned} x_3 &= (x_1 + x_2)/2 \\ y_3 &= (y_1 + y_2)/2 \\ z_3 &= (z_1 + z_2)/2 \end{aligned} \tag{5.7}$$

The direction vector (a_{03}, b_{03}, c_{03}) between this point and the sphere centre *C* is computed and the result is:

$$\begin{pmatrix} a_{03} \\ b_{03} \\ c_{03} \end{pmatrix} = \begin{pmatrix} x_3 \\ y_3 \\ z_3 \end{pmatrix} - \begin{pmatrix} x_0 \\ y_0 \\ z_0 \end{pmatrix} \tag{5.8}$$

This direction vector is normalized as before and the new sphere start point *T* is obtained a distance equal to the sphere radius from the sphere centre in the direction of (a_{03}, b_{03}, c_{03}):

$$\begin{pmatrix} x_4 \\ y_4 \\ z_4 \end{pmatrix} = \begin{pmatrix} x_0 \\ y_0 \\ z_0 \end{pmatrix} + r \begin{pmatrix} a_{03} \\ b_{03} \\ c_{03} \end{pmatrix} \tag{5.9}$$

5.2.3 Candidate Nodes for Inclusion Identified

The algorithm searches for octree neighbours of the nodes included during the previous iteration round. The neighbour search algorithm (Section 4.2) is used during this step.

5.2.4 Testing of Candidate Nodes

The points in the candidate nodes are tested (based on the parameter values of the most recently fitted geometric entity) by using a user specified error tolerance to determine if they belong to the recently fitted entity. A point is included if the shortest distance from the point to the computed geometric entity surface is smaller than or equal to the error tolerance. This error tolerance is of the same order as the accuracy of the point-measuring device used. It is sensible for the user to provide an error tolerance slightly larger than the accuracy of the measuring device, otherwise good points might be excluded from the fit during the entity grow phase. The entity boundary may not be computed accurately if the error tolerance is set smaller than the accuracy of the point-measuring device.

Node tolerances are now computed for each of the candidate nodes. The node tolerance is used to verify whether points from a specific node can be included in the geometric entity fit. The absolute distance from each point in a node to the entity surface is computed. Points in this node will be included if they satisfy the error tolerance. The node tolerance is the number of points that will be included divided by the total number of points in this specific node. Points from this specific node will be used in the entity fit if this fraction is larger than the specified node tolerance. The node tolerance is used to find the boundary of the geometric entity. Nodes at the boundary of the entity will contain points that belong to the next entity. Nodes will be regarded as boundary nodes if the number of points belonging to the recently fitted entity divided by the total number of points in the node is smaller than the node tolerance. Therefore the entity may be grown over its boundary during the entity grow phase if the node tolerance value is too close to zero and visa versa.

5.2.5 Recursive Inclusion of Nodes

Neighbour nodes are now found for nodes that were included during the previous phase. This process (steps in section 5.2.3 and 5.2.4) is repeated until no more nodes can be found that satisfy the node tolerance. This recursive manner of including nodes is more efficient (with respect to time spend to grow the entity) than one where the entity parameters are recalculated after each neighbour node search round.

5.2.6 Fitting of Entity

New entity parameters are now obtained by refitting the entity to the currently included points with the use of least squares as explained in Chapter 3. Points that belong to nodes that are considered as boundary nodes (as a result of their node tolerance value) are not included in the fit. All points and nodes that were considered part of the entity is now rechecked to determine if they are still part of the entity or not. The points and nodes will then be included/excluded accordingly during the next entity fit.

5.2.7 Recursive Fitting of Entity

The grow algorithm returns to the step where new candidate nodes are tested for inclusion (Section 5.2.3). It repeats the recursive testing and inclusion of nodes. The entity is refitted in the next step. The grow algorithm checks after each entity fit step if new nodes were added during the current iteration. The boundary nodes of the entity are found if no more nodes can be included that belong to the entity. This is the stop criterion for the recursive part. The grow algorithm continues by fitting a final entity described in the next section.

5.2.8 Fitting of Final Entity

In this step the points in all boundary nodes, as well as their neighbours, are ignored when the final entity is fitted, because these nodes are more likely to contain points that do not belong to the entity. These points are ignored to ensure that the final entity parameter values will not be influenced by points that do not belong to the entity.

5.2.9 Finding the Actual Entity Boundary

This is the last step of the grow algorithm. All the points in the nodes excluded from the final entity fit (Section 5.2.8) are now re-evaluated according to the final fit parameter values, using the boundary error tolerance, because some of these points may be part of the entity.

The distance from each of these points to the calculated entity surface is computed. A specific point is added to the entity as a boundary point if the absolute distance is smaller than the boundary error tolerance. The boundary error tolerance may be equal to or smaller than the error tolerance. Points belonging to the actual entity may be discarded if the boundary error tolerance value is too tight. The opposite will happen if the boundary error tolerance is too lenient.

5.2.10 User Evaluation of Fitted Entity

A point file that includes all the points that belong to the entity is created after the entity boundary is determined. The entity points can then be displayed in AutoCAD in a user-selected layer on top of the original point cloud. This enables the user to evaluate the extracted entity and to extract the entity once more if necessary.

5.3 Summary of User Input Parameters

This section is included to give the user an overall grasp on the specific input that he/she has to supply to the entity extraction algorithm. The grow algorithm requires the following user input:

- the pitch (average distance between the measured data points) – Section 5.2.1
- the node size of the octree as a multiplication factor of the pitch – Section 5.2.1
- start point(s) for the entity – Section 5.2.2
- the error tolerance of the measured points – Section 5.2.4
- the node tolerance – Section 5.2.5
- the boundary error tolerance – Section 5.2.9

5.4 Plane Extraction Example

The entity growing process is explained in more detail with the following example where a flat plane is fitted. It is only showed for the two-dimensional case for clarity. Figure 5.3 shows part of a plane divided into twenty-five equal sized octree nodes. The octree node that contains the initial plane start point is hatched in two directions.

The grow algorithm continues now by adding nodes (one-way hatched nodes in Figure 5.4) that satisfy the node tolerance as described in Sections 5.2.3 - 5.2.5. The plane is now refitted as described in Section 5.2.6.

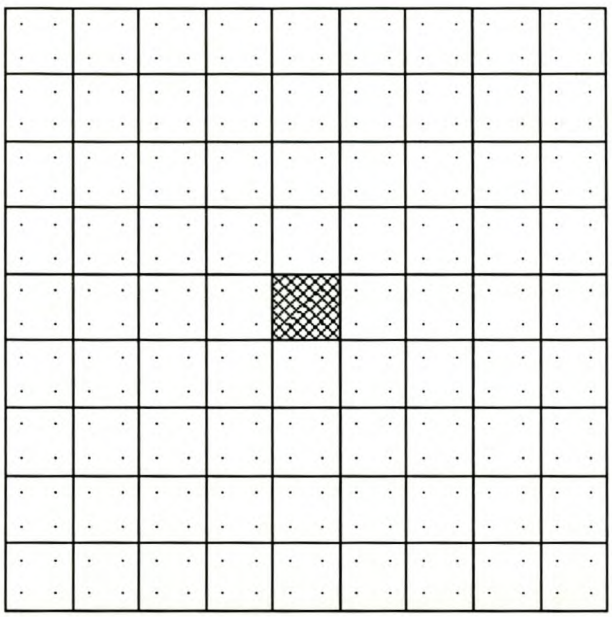

Figure 5.3 Part of a plane where the node that contains the start point is hatched

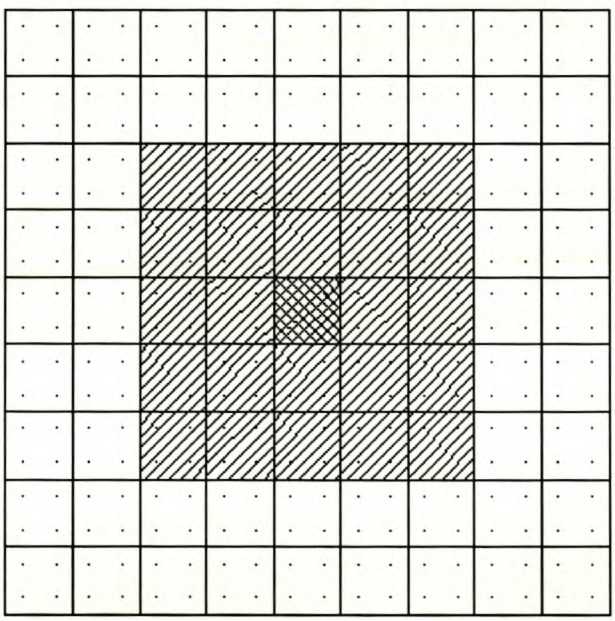

Figure 5.4 Initial node with added neighbour nodes

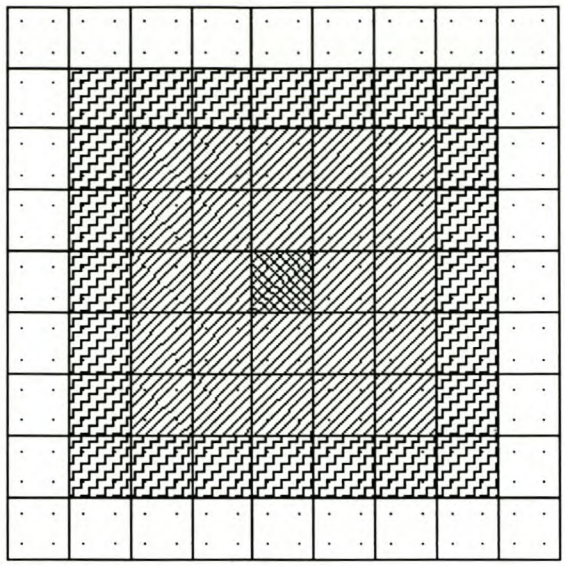

Figure 5.5 More neighbour nodes added after the plane was refitted

Figure 5.5 shows the next round where more nodes are added to the entity as described in Section 5.2.7. This process continues until the plane boundary is found. The grow process stop here as no more nodes can be found that satisfy the node tolerance as described Section 5.2.7. The plane boundary is shown as triangle hatched nodes in Figure 5.6. The algorithm identifies the boundary nodes and their neighbours in the next step (Section 5.2.8) as shown in Figure 5.7 and uses only the 45° hatched and double hatched nodes to compute the final plane parameters. The last step (section 5.2.9) searches for all the points in the non-hatched nodes (Figure 5.7) that satisfy the boundary error tolerance. This is how the actually boundary is found. The entity grow algorithm is explained in detail in Appendix A with the aid of flow charts.

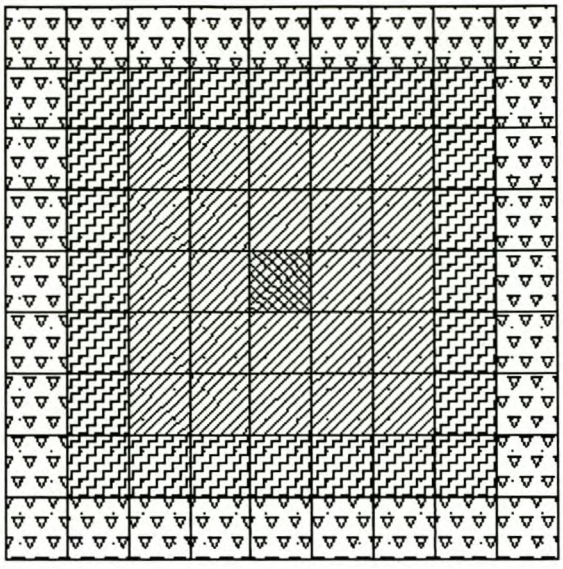

Figure 5.6 Triangle hatched nodes are the found boundary of the plane

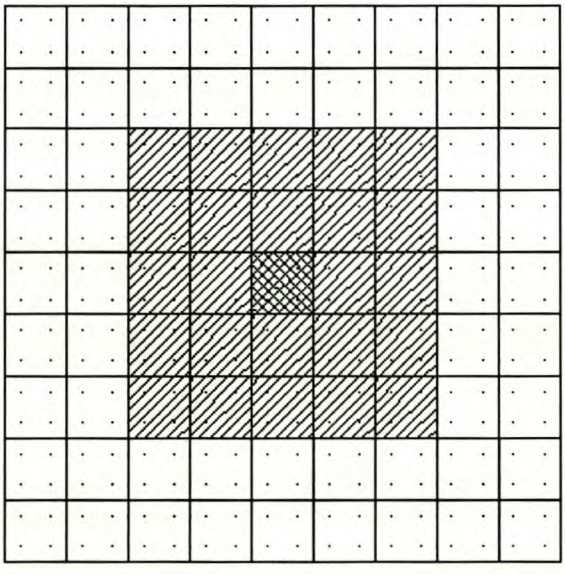

Figure 5.7 Nodes selected to fit the final plane parameters

5.5 Octree Node Weights

Weights were assigned to octree nodes during algorithm development. The idea was that nodes selected at the start of the search process had a better chance to contain points that will be part of the entity than nodes included later in the search progress. Nodes selected during the first step would have a weight of one. As soon as a new cycle (steps a to c) was started one was added to the weight of the nodes already

included. Therefore, the node weight will increase linearly from the boundary towards the initial entity start node. Points were included in the entity fit as many times as the weight of the node they belonged to. This scheme assured that points belonging to the first nodes selected had a bigger influence on the final entity fitted than points that belonged to nodes closer to the boundary of the entity.

This process had more disadvantages than advantages. In some cases it found the entity parameters a bit better than the method that did not use the point weights. The main disadvantage was that it made the fit process very slow when big entities were fitted. The reader will note that if an entity consists of 10000 points and about 20 grow levels, some points will be included up to 20 times in the fit procedure. This is the reason for the increase in fitting time. Another disadvantage was that an uncertainty still existed as to how well the initial entity start point was selected by the user. Nodes near the boundary of the entity will have a larger weight than other nodes at the opposite boundary if the user selected a start point far from the centre of the entity. This problem is shown in Figure 5.8. The user selected a start point for the grow algorithm at the node with weight equal to six. This is the weight assigned to this node at the end of the grow process. Figure 5.8 illustrates points in boundary nodes with levels of three, two and one respectively. This node weight scheme was discarded because of these two reasons.

All the points selected are tested after each entity fit stage to see if it satisfy the error tolerance. Points that violate the error tolerance are not used during the next fit. These points are again tested for inclusion after the next entity parameters were obtained. Another weight scheme was tried where nodes that contain points that are not excluded from any fit during the grow process carries a larger weight than nodes where some of the points are excluded during some entity fit stages. The reasoning behind this scheme was that the chances are better for points that are always included in the fit stages to belong to the final entity. This scheme did not prove to be more successful than using a grow algorithm without it. It also had the earlier mentioned disadvantage of increasing the entity extraction time. Therefore this scheme was discarded.

2	2	2	2	2	2	2	2	1
3	3	3	3	3	3	3	2	1
3	4	4	4	4	4	3	2	1
3	4	5	5	5	4	3	2	1
3	4	5	6	5	4	3	2	1
3	4	5	5	5	4	3	2	1
3	4	4	4	4	4	3	2	1
3	3	3	3	3	3	3	2	1
2	2	2	2	2	2	2	2	1

Figure 5.8 Problem with weights if a start point far from the entity centre is selected

6 Validation of Entity Search Algorithm

This chapter describes testing of the entity extraction algorithm. Different entities are created and the extraction algorithm is employed to extract these entities. The extraction algorithm is then evaluated on how well the entity was fitted. Computer generated point clouds are used for the case studies. The algorithm is also used to extract entities from coordinate measuring machine created point clouds for completeness.

6.1 Design of Experiments

Design of experiments [O'Connor, 1991] was used to test the entity extraction algorithm. It forces the developer to test the algorithm using many test cases with a wide spectrum of parameter values. This type of test is performed to inform the user of parameters that will have a big influence on the final result. The outcome is to give the user a guideline on how to choose input values (described in Chapter 5) for the entity search algorithm. Implications of each user input value will be described. Thirty-two test cases were done for each of the three (planes, cylinders and spheres) geometric entity extraction methods.

Six variables were selected for the plane and sphere extraction case studies. Seven variables were identified for the cylinder extraction cases. This means that 64 test case combinations exist for each of the plane and sphere extraction methods to evaluate all of the variable combinations. In the case of the cylinder extraction method, this number increases to 128 test cases. The computer software Statistica [Version 5.1, 1997] was used to reduce the number of experiments that had to be carried out for each of the extraction methods. The user tells the program how many variables are involved and how many experiments he/she wants to perform. The number of experiments can be 8, 16, 32 and 64 for level six (six variables) experiments and up to 128 for level seven (seven variables) experiments. Statistica returns the influences of the number of experiments on the final experiment results. The chances of combinations of variables aliasing with other combinations of variables increase as the number of experiments is made smaller. Aliasing means that

the effect of one combination of variables on the outcome of the experiment is indistinguishable from another combination of variables.

It was decided to do 32 experiments for each of the three extraction methods. Statistica warned the user about lower order aliasing in the case of the cylinder experiments. These combinations of variables did not make any sense at all and therefore it was decided to go ahead with 32 experiments for the cylinder extraction case. Aliasing did not occur for the 32 experiments chosen for the plane and sphere extraction cases.

Design of experiments works on the following basis shown for a case of three independent variables. Possible high and low values are assigned to each of the chosen variables. Therefore, each variable will have a high and a low value for the experiments. Table 6.1 shows the combinations of variables for the eight case studies to be done for a level three experiment as well as the dependant variable values of the case study with these combinations.

Experiment #	Independent Variable A	Independent Variable B	Independent Variable C	Dependent Variable 1	Dependent Variable 2
1	-1	-1	-1	15	5
2	-1	-1	1	8	3
3	-1	1	-1	10	2
4	-1	1	1	12	6
5	1	-1	-1	19	1
6	1	-1	1	9	5
7	1	1	-1	13	4
8	1	1	1	11	3

Table 6.1 Example combinations of independent variables and the results of these combinations on the dependent variables

A high variable value is related to a positive one and a low variable value to a negative one. The values for independent variable *A* are multiplied with the values of dependent variable 1 and these values are added to see what influence independent

variable *A* has on dependent variable 1. This calculation is repeated for all the independent and dependent variables. Results are shown in Table 6.2.

	Influence on Dependent Variable 1	Influence on Dependent Variable 2
Variable A	7	-3
Variable B	-5	1
Variable C	-17	5

Table 6.2 Results of example experiments

From Table 6.2 it can be seen that independent variable *C* has the biggest influence on dependent variables 1 and 2. An influence of zero means that an independent variable has no influence on a specific dependent variable. This experiment shows the user that special care must be taken when setting values for independent variable *C*.

6.2 Variables and Measured Values

This section explains the motives for choosing certain variables for the case study experiments. The reasons for specific results measured are also explained.

6.2.1 Experiment Design Variables

The following design variables were chosen for the case study experiments:

- entity length divided by point pitch (planes and cylinders) – l/pp
- entity radius divided by point pitch (spheres and cylinders) – r/pp
- length divided by radius (only cylinders) – l/r
- fit accuracy divided by data accuracy (planes, cylinders and spheres) – fa/da
- node size divided by point pitch (planes, cylinders and spheres) – nds/pp
- node tolerance (planes, cylinders and spheres) – ntol
- boundary node accuracy divided by data accuracy (planes, cylinders and spheres) – bna/da
- data accuracy (planes, cylinders and spheres) – da.

The reader may note that eight design variables relate to the cylinder test cases. The larger of the first two mentioned bullet values was discarded for the cylinder test cases. This gives seven design variables for the cylinder test cases and six design variables for the plane and sphere test cases respectively. An explanation of the importance of the design variables follows.

The length or radius divided by the point pitch will relate somehow to how well the entity will be fitted. An entity will probably be fitted more accurately as this variable value becomes larger. The length divided by the radius is proposed to test the ability of the entity extraction algorithm to extract long slender cylinders and circular disks. The fit accuracy divided by the data accuracy will determine the accuracy to which the entity will be fitted. A ratio of one will ensure a more accurate fit than a larger ratio, but the fitting process may give inaccurate results regarding points that must be included or excluded from the entity fit. It does not make any sense to set this ratio smaller than one.

The node size divided by the point pitch has a big influence on the octree data structure build time. It consumes more computer time to built an octree if this ratio is small because the octree nodes are refined further. The entity boundary might be inaccurately computed if this ratio is too large. The reason is that the node size determines the search steps. The node tolerance is the number of points included in a node divided by all the points in that node as explained in Chapter 5. This ratio is therefore between zero and one. The entity will be fitted more accurately if this ratio is close to one, but the fit process may find the entity boundary incorrectly as compared to when the ratio is smaller.

The boundary node accuracy determines which points in the boundary nodes will be added to the entity after the final entity was fitted. The boundary accuracy divided by the data accuracy is chosen as a ratio between zero and one. The boundary of the entity may be found incorrectly if this ratio is too close to either extreme value. The boundary of the entity may be larger if this ratio is close to one, or smaller than the actual boundary if the ratio is close to zero.

The data accuracy will obviously play a role in the final fit accuracy of the entity. This variable value is something the user of the algorithm can do nothing about. It is determined by the point cloud-scanning machine.

The high and low values for the different variables are shown in Table 6.3-6.5 for the plane, sphere and cylinder test cases. These values were chosen to be realistic as values that would be chosen by the user during the fit procedure. Therefore, extreme values were avoided.

PLANES	Low value	High value
l/pp	20	200
fa/da	1	3
nds/pp	3	5
ntol	0.5	0.7
bna/da	0.1	0.5
da	10 μm	50 μm

Table 6.3 High and low values for the design parameters of the plane test cases

SPHERES	Low value	High value
r/pp	10	100
fa/da	1	3
nds/pp	2	3
ntol	0.5	0.7
bna/da	0.1	0.5
da	10 μm	50 μm

Table 6.4 High and low values for the design parameters of the sphere test cases

CYLINDERS	Low value	High value
smallest ratio of l/pp or r/pp	10	100
l/r	0.5	4
fa/da	1	3
nds/pp	3	4
ntol	0.5	0.7
bna/da	0.1	0.5
da	10 μm	50 μm

Table 6.5 High and low values for the design parameters of the cylinder test cases

6.2.2 Results Measured

This paragraph explains the reasons for the selection of the measured results for the case study experiments. The following common results for all entity test cases (planes, spheres and cylinders) were measured.

- octree data structure creation time in seconds – oct_time
- entity extraction time in seconds – ent_time
- average point distance difference to entity surface – avep_ddiff (explained on next page)
- number of correct points included divided by the number of all points belonging to the entity – CPI
- number of wrong points included divided by the number of all points belonging to the entity – WPI

The octree data structure creation time as well as the entity extraction time is directly related to the node size. If the node size is made larger, these two processes will be executed quicker as the refinement of the blocks stops earlier and the search step is larger. Therefore, the design parameter influences on these two results will not be calculated. The times are measured to compare them between the same entities extracted on different computers.

The value for avep_ddiff is calculated as follows: The actual entity is fitted to all the points belonging to the entity. These points are labelled when the point cloud was generated. The entity parameter values are then known. The entity is then grown from the start point and the extracted entity parameters are found. The average absolute distance of all the points belonging to the entity to the actual entity surface is then calculated. The average absolute distance of the same points to the extracted entity surface is then calculated. The avep_ddiff is the difference between the second mentioned average absolute distance and the first mentioned absolute average distance. This difference is equal to zero if all the points belonging to the entity were used to fit the final extracted entity. The extracted entity accuracy becomes worse as this difference drifts away from zero.

The ratio CPI will be between zero and one. The entity extraction is more successful as this ratio becomes closer to one. The ratio WPI will be bigger or equal to zero. The entity extraction accuracy decreases as this ratio drifts away from zero.

The following results are measured regarding the extraction of planes.

- angle between actual plane's normal vector and the normal vector of the extracted plane – angle
- centroid offset difference – coff_diff

The actual plane is fitted to all the data points that belong to the plane. The extracted plane is grown from a selected start point. Both normal vectors are computed. The angle between these two vectors is calculated using the dot product. The accuracy of the extracted entity improves as the angle becomes closer to zero. The centroid of all the points belonging to the actual entity is computed as well as the centroid of all the data points included in the final extracted plane. The difference between these two values is computed. The extracted plane becomes less accurate as the difference between these two values becomes bigger.

The following results are measured regarding the extraction of spheres.

- difference between the centre of the actual sphere and that of the extracted sphere – cent_diff
- difference between the radius of the actual sphere and the radius of the extracted sphere – delta_r

The extracted sphere's accuracy improves as these two differences tend to zero.

The following results are measured regarding the extraction of cylinders.

- angle between the axis vector of the actual cylinder and that of the extracted cylinder – angle
- difference between the radius of the actual cylinder and that of the extracted cylinder – delta_r
- minimum distance between the axis vector of the actual cylinder and the axis vector of the extracted cylinder – min_axis_diff

The accuracy of the extracted cylinder improves as these three mentioned results tend to zero.

6.3 Case Study Point Cloud

The point cloud chosen for performing the case study had all three types of entities included. It consists of four planes, four fillets and an eighth of a sphere. The sizes of the entities, the average point pitch, the orientation of the point cloud and the volumetric error assigned to each point can be altered by the user. The volumetric error is computed by multiplying the maximum tolerated error supplied by the user with a normal distribution with mean deviation of zero and standard deviation of 0.25. If the absolute value of the volumetric error is larger than the error supplied by the user, it is assigned the maximum error tolerance. The entities flow into each other with smooth boundaries. This property is a challenge for the entity extraction algorithm as it makes the borders of the entities more difficult to determine. Figure

6.1 shows the computer generated point cloud with the three entities used for the case studies.

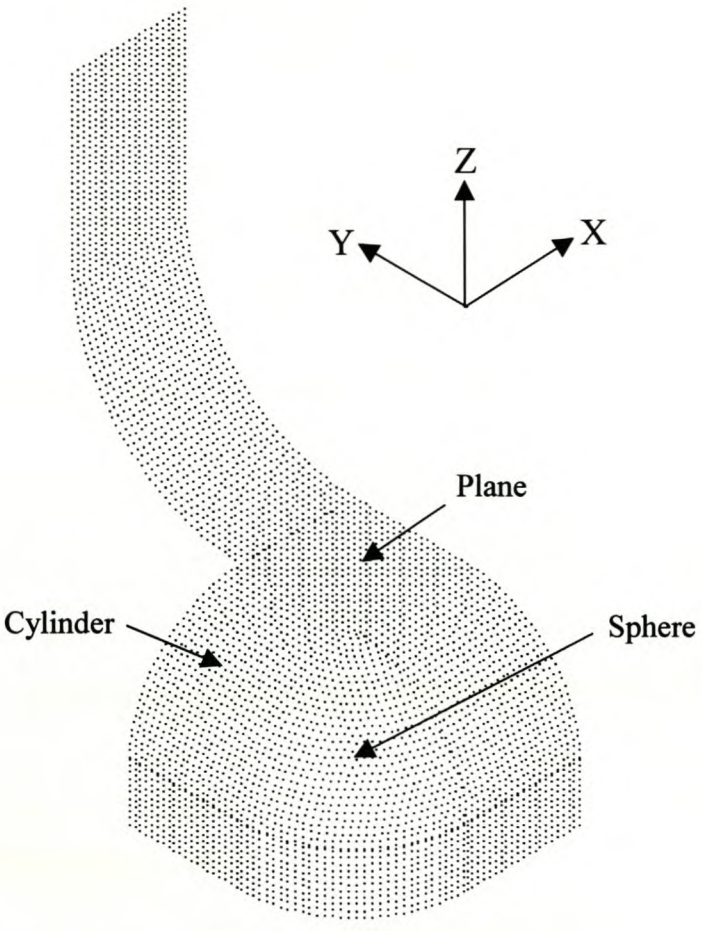

Figure 6.1 Computer generated point cloud consisting of different entities

The sizes of the three entities are varied to suit the high and low values of the different design variables. The middle of the plane shown in Figure 6.1 is the origin of the point cloud. The point cloud is rotated around the origin so that the normal vector of the plane lies at an angle of 66.42 degrees to the x-axis, 45.57 degrees to the y-axis and 53.73 degrees to the z-axis. This corresponds to the direction cosines of:

$$\begin{aligned} a &= 0.4 \\ b &= 0.7 \\ c &= \sqrt{1-0.4^2-0.7^2} = 0.591608 \end{aligned} \tag{6.1}$$

The direction vector of the plane's normal vector and the cylinder axis vector as well as the centroid of the plane, the centre of the sphere and a point on the cylinder axis were not considered as design variables of the case study. The position and the rotation of each entity will be correctly computed if the correct points are used to compute the entity parameter values. This also decreased the number of experiments that had to be carried out.

6.4 Results and Guidelines

The results of the design of experiments case study are presented in this paragraph. Guidelines for choosing input values for the extraction algorithm are also presented. A comparison between the computer time spent to build the octree data structure as well as the time spent to extract the entity by three different computers is presented.

6.4.1 Results of Case Studies

Tables 6.6 – 6.14 show the results for the case studies done on the plane, sphere and cylinder. Extreme values are highlighted in the tables.

6.4.1.1 Plane results

Table 6.6 shows the high and low values for the different design variables as discussed earlier. The measured results obtained by using these design variable values are shown in Table 6.7. Extreme values are highlighted.

Plane #	l/pp	fa/da	nds/pp	ntol	bna/da	da [μm]
1	20	1	3	0.5	0.1	10
2	200	1	3	0.5	0.1	50
3	20	3	3	0.5	0.1	50
4	200	3	3	0.5	0.1	10
5	20	1	5	0.5	0.1	50
6	200	1	5	0.5	0.1	10
7	20	3	5	0.5	0.1	10
8	200	3	5	0.5	0.1	50
9	20	1	3	0.7	0.1	50
10	200	1	3	0.7	0.1	10
11	20	3	3	0.7	0.1	10
12	200	3	3	0.7	0.1	50
13	20	1	5	0.7	0.1	10
14	200	1	5	0.7	0.1	50
15	20	3	5	0.7	0.1	50
16	200	3	5	0.7	0.1	10
17	20	1	3	0.5	0.5	50
18	200	1	3	0.5	0.5	10
19	20	3	3	0.5	0.5	10
20	200	3	3	0.5	0.5	50
21	20	1	5	0.5	0.5	10
22	200	1	5	0.5	0.5	50
23	20	3	5	0.5	0.5	50
24	200	3	5	0.5	0.5	10
25	20	1	3	0.7	0.5	10
26	200	1	3	0.7	0.5	50
27	20	3	3	0.7	0.5	50
28	200	3	3	0.7	0.5	10
29	20	1	5	0.7	0.5	50
30	200	1	5	0.7	0.5	10
31	20	3	5	0.7	0.5	10
32	200	3	5	0.7	0.5	50

Table 6.6 High and low values for the thirty-two different plane design variables

Plane#	angle [°]	coff_diff [mm]	avep_ddiff [mm]	CPI	WPI
1	0.00250	1.9050	1.39E-05	0.7143	0.0045
2	0.00004	2.9516	2.82E-07	0.9426	0.0000
3	0.00517	0.6014	3.49E-06	0.9161	0.0181
4	0.00001	2.9366	1.38E-08	0.9409	0.0000
5	0.01045	1.5678	3.28E-05	0.7256	0.0340
6	0.00002	5.9318	2.90E-08	0.8826	0.0000
7	0.00459	1.5630	5.85E-05	0.7234	0.0045
8	0.00013	5.0591	1.38E-06	0.9046	0.0000
9	0.00333	1.6246	1.69E-05	0.7279	0.0091
10	0.00002	4.0191	1.61E-07	0.9306	0.0000
11	0.00157	1.5080	-4.50E-06	0.6984	0.0045
12	0.00010	2.0310	6.32E-07	0.9514	0.0000
13	0.00508	1.2419	4.18E-05	0.6508	0.0045
14	0.00005	6.3322	-1.07E-06	0.8729	0.0000
15	0.01467	1.6007	1.07E-04	0.7279	0.0136
16	0.00001	6.3383	-2.21E-07	0.8720	0.0000
17	0.00423	1.5730	4.25E-06	0.9773	0.0998
18	0.00002	3.0568	2.39E-07	0.9956	0.0000
19	0.00151	1.5630	-6.53E-06	0.9819	0.0249
20	0.00007	1.9789	-5.59E-08	0.9981	0.0001
21	0.00209	1.9050	3.63E-06	0.9819	0.0340
22	0.00008	5.9099	-2.80E-07	0.9920	0.0003
23	0.00145	0.6012	1.91E-06	0.9909	0.0975
24	0.00001	5.9358	5.60E-10	0.9932	0.0000
25	0.00157	1.2419	3.67E-06	0.9773	0.0272
26	0.00019	4.0689	2.52E-06	0.9953	0.0005
27	0.00962	1.6164	1.31E-05	0.9728	0.0703
28	0.00003	4.0191	1.45E-07	0.9948	0.0000
29	0.00836	1.4596	4.44E-05	0.9841	0.0862
30	0.00002	6.3370	4.61E-08	0.9909	0.0000
31	0.00383	1.5082	2.71E-05	0.9705	0.0181
32	0.00017	5.7416	1.65E-06	0.9928	0.0002

Table 6.7 Results for the thirty-two planes extracted

	angle [°]	coff_diff [mm]	avep_ddiff [mm]	CPI	WPI
l/pp	-7.91E-02	4.96E+01	-3.56E-04	1.53E+00	-5.50E-01
fa/da	4.86E-03	-6.52E+00	4.03E-05	2.88E-01	-4.81E-02
nds/pp	2.10E-02	2.23E+01	2.71E-04	-4.59E-01	3.39E-02
ntol	1.63E-02	5.65E+00	1.40E-04	-3.51E-01	-8.36E-02
bna/da	-1.45E-02	1.30E+00	-1.75E-04	2.61E+00	3.66E-01
da	3.52E-02	-6.29E+00	9.08E-05	3.73E-01	3.07E-01

Table 6.8 Influences of design variables on the measured results for planes

Table 6.8 shows the influences of the design variables on the measured results. The design variable that had the biggest influence in each result category is highlighted. The plane length divided by the point pitch had the biggest influence in four of the result categories. This tendency is expected as the accuracy of the extracted entity improves as more points are included in the entity fit. The boundary node accuracy divided by the data accuracy determined the accuracy of the entity boundary prediction.

The best extraction results were obtained for test cases 3 and 25 (small 20mm x 20mm planes) and for test cases 18 and 20 (large 200mm x 200mm planes). Pictures of fitted planes are presented in Appendix B.

6.4.1.2 Sphere results

Table 6.9 shows the high and low values for the different design variables as discussed in section 6.2.1. The measured results obtained by using these design variable values are shown in Table 6.10. Extreme values are highlighted. Table 6.11 shows the measured results and the design variable that has the biggest influence on each result is highlighted. The sphere radius divided by the point pitch had the biggest influence on almost all of the measured results. This corresponds well with the tendency recognized in the plane cases. The data accuracy plays a minor role in the accuracy of the extracted sphere.

Sphere #	r/pp	fa/da	nds/pp	ntol	bna/da	da
1	10	1	2	0.5	0.1	0.01
2	100	1	2	0.5	0.1	0.05
3	10	3	2	0.5	0.1	0.05
4	100	3	2	0.5	0.1	0.01
5	10	1	3	0.5	0.1	0.05
6	100	1	3	0.5	0.1	0.01
7	10	3	3	0.5	0.1	0.01
8	100	3	3	0.5	0.1	0.05
9	10	1	2	0.7	0.1	0.05
10	100	1	2	0.7	0.1	0.01
11	10	3	2	0.7	0.1	0.01
12	100	3	2	0.7	0.1	0.05
13	10	1	3	0.7	0.1	0.01
14	100	1	3	0.7	0.1	0.05
15	10	3	3	0.7	0.1	0.05
16	100	3	3	0.7	0.1	0.01
17	10	1	2	0.5	0.5	0.05
18	100	1	2	0.5	0.5	0.01
19	10	3	2	0.5	0.5	0.01
20	100	3	2	0.5	0.5	0.05
21	10	1	3	0.5	0.5	0.01
22	100	1	3	0.5	0.5	0.05
23	10	3	3	0.5	0.5	0.05
24	100	3	3	0.5	0.5	0.01
25	10	1	2	0.7	0.5	0.01
26	100	1	2	0.7	0.5	0.05
27	10	3	2	0.7	0.5	0.05
28	100	3	2	0.7	0.5	0.01
29	10	1	3	0.7	0.5	0.05
30	100	1	3	0.7	0.5	0.01
31	10	3	3	0.7	0.5	0.01
32	100	3	3	0.7	0.5	0.05

Table 6.9 High and low values for the thirty-two different sphere design variables

Sphere#	cent_diff [mm]	delta_r [mm]	avep_ddiff [mm]	CPI	WPI
1	0.003896	0.0034456	9.46E-05	0.6769	0.0000
2	0.000545	0.0004913	-1.36E-07	0.9928	0.0106
3	0.056870	0.0513489	1.32E-03	0.9949	0.0923
4	0.000028	0.0000230	1.41E-08	0.9766	0.0000
5	0.022098	0.0197779	2.48E-04	0.6462	0.0000
6	0.000076	0.0000526	3.09E-08	0.9221	0.0000
7	0.001574	0.0007121	2.99E-05	0.5487	0.0000
8	0.001352	0.0012077	9.73E-07	0.9857	0.0117
9	0.011550	0.0087960	1.67E-04	0.7795	0.0000
10	0.000024	0.0000233	5.28E-09	0.9542	0.0000
11	0.006905	0.0064095	2.08E-04	0.6051	0.0000
12	0.015861	0.0144856	9.17E-05	1.0000	0.0434
13	0.024750	0.0244718	2.22E-03	0.2872	0.0000
14	0.000116	0.0000949	5.58E-08	0.9404	0.0072
15	0.014193	0.0067007	4.82E-04	0.6051	0.0000
16	0.000147	0.0001234	-4.62E-08	0.9211	0.0000
17	0.006569	0.0005495	1.11E-04	0.9641	0.0103
18	0.000052	0.0000418	5.23E-09	0.9975	0.0009
19	0.001854	0.0014953	4.20E-06	0.9846	0.0000
20	0.025437	0.0233190	2.35E-04	1.0000	0.0590
21	0.009214	0.0088465	6.34E-04	0.8769	0.0000
22	0.000691	0.0005879	1.43E-07	0.9971	0.0293
23	0.037176	0.0336586	7.41E-04	0.9846	0.0667
24	0.000113	0.0000967	6.20E-08	0.9964	0.0009
25	0.003959	0.0035249	1.14E-04	0.9436	0.0000
26	0.000545	0.0004940	-8.52E-08	0.9994	0.0288
27	0.038150	0.0344921	8.61E-04	1.0000	0.1026
28	0.000032	0.0000194	1.17E-08	0.9981	0.0006
29	0.013579	0.0105244	1.24E-04	0.9744	0.0000
30	0.000074	0.0000281	-2.92E-08	0.9944	0.0006
31	0.034146	0.0331989	3.14E-03	0.5744	0.0000
32	0.000594	0.0004875	4.78E-07	0.9987	0.0296

Table 6.10 Results for the thirty-two spheres extracted

	cent_diff [mm]	delta_r [mm]	avep_ddiff [mm]	CPI	WPI
r/pp	-2.41E-01	-2.06E-01	-1.02E-02	3.23E+00	-4.92E-02
fa/da	1.37E-01	1.26E-01	3.40E-03	2.27E-01	3.19E-01
nds/pp	-1.24E-02	-8.39E-03	4.41E-03	-1.61E+00	-2.03E-01
ntol	-2.92E-03	-1.78E-03	3.99E-03	-9.70E-01	-6.90E-02
bna/da	1.22E-02	1.32E-02	1.11E-03	2.45E+00	1.64E-01
da	1.58E-01	1.25E-01	-2.06E-03	1.61E+00	4.88E-01

Table 6.11 Influences of design variables on the measured results for spheres

The boundary node accuracy had the second biggest influence on the result CPI. This result is also expected as the boundary points are added if their distance to the entity surface satisfies the boundary error tolerance. The best extraction results were obtained for test cases 17 and 19 (small 10mm radius spheres) and for test cases 4, 28 and 30 (large 200mm radius spheres). Pictures of fitted spheres are presented in Appendix B.

6.4.1.3 Cylinder results

Table 6.12 shows the high and low values for the different design variables as discussed in section 6.2.1. The measured results obtained by using these design variable values are shown in Table 6.13. Extreme values are highlighted. Table 6.14 shows the measured results and the design variable that has the biggest influence on each result is highlighted. The minimum of the cylinder length or cylinder radius divided by the point pitch had the biggest influence on almost all of the measured results. This corresponds well with the tendency recognized for planes and spheres. The accuracy of the extracted cylinder improves as the point cloud becomes denser.

The data accuracy plays a role in three of the measured results. The accuracy of the extracted cylinder improves as the point cloud becomes more accurate. The boundary node accuracy has a big influence on the result CPI. This result is also expected, as before, because the boundary points are added if their distance to the entity surface satisfies the boundary error tolerance.

Cyl #	l/r	l/pp or r/pp	fa/da	nds/pp	ntol	bna/da	da
1	0.5	10	1	3	0.5	0.5	0.05
2	4	10	1	3	0.5	0.1	0.01
3	0.5	100	1	3	0.5	0.1	0.01
4	4	100	1	3	0.5	0.5	0.05
5	0.5	10	3	3	0.5	0.1	0.05
6	4	10	3	3	0.5	0.5	0.01
7	0.5	100	3	3	0.5	0.5	0.01
8	4	100	3	3	0.5	0.1	0.05
9	0.5	10	1	4	0.5	0.1	0.01
10	4	10	1	4	0.5	0.5	0.05
11	0.5	100	1	4	0.5	0.5	0.05
12	4	100	1	4	0.5	0.1	0.01
13	0.5	10	3	4	0.5	0.5	0.01
14	4	10	3	4	0.5	0.1	0.05
15	0.5	100	3	4	0.5	0.1	0.05
16	4	100	3	4	0.5	0.5	0.01
17	0.5	10	1	3	0.7	0.5	0.01
18	4	10	1	3	0.7	0.1	0.05
19	0.5	100	1	3	0.7	0.1	0.05
20	4	100	1	3	0.7	0.5	0.01
21	0.5	10	3	3	0.7	0.1	0.01
22	4	10	3	3	0.7	0.5	0.05
23	0.5	100	3	3	0.7	0.5	0.05
24	4	100	3	3	0.7	0.1	0.01
25	0.5	10	1	4	0.7	0.1	0.05
26	4	10	1	4	0.7	0.5	0.01
27	0.5	100	1	4	0.7	0.5	0.01
28	4	100	1	4	0.7	0.1	0.05
29	0.5	10	3	4	0.7	0.5	0.05
30	4	10	3	4	0.7	0.1	0.01
31	0.5	100	3	4	0.7	0.1	0.01
32	4	100	3	4	0.7	0.5	0.05

Table 6.12 High and low values for the thirty-two different cylinder design variables

Cyl #	angle [°]	delta_r [mm]	min_axis_diff [mm]	avep_ddiff [mm]	CPI	WPI
1	0.009510	0.00798	0.0066552	5.794E-05	0.9835	0.1047
2	0.000502	0.00156	0.0012134	1.748E-05	0.6901	0.0000
3	0.000015	0.00001	0.0000041	2.341E-08	0.9909	0.0035
4	0.000004	0.00009	0.0000985	-1.622E-08	0.9991	0.0155
5	0.020714	0.00677	0.0075564	1.352E-04	0.8375	0.0193
6	0.002475	0.00031	0.0003743	4.501E-05	0.9756	0.0029
7	0.000111	0.00039	0.0000031	3.297E-07	1.0000	0.0183
8	0.000017	0.00523	0.0047556	1.013E-05	0.9999	0.0168
9	0.001995	0.00243	0.0023300	3.870E-05	0.6804	0.0028
10	0.003294	0.00610	0.0053070	8.764E-05	0.9799	0.0029
11	0.000075	0.00011	0.0000288	1.109E-07	0.9992	0.0423
12	0.000006	0.00009	0.0000973	1.193E-07	0.9461	0.0000
13	0.009350	0.00405	0.0041264	7.291E-05	0.9807	0.0138
14	0.003774	0.00192	0.0020879	1.479E-05	0.7633	0.0057
15	0.001744	0.00429	0.0013290	2.721E-05	0.9997	0.0504
16	0.000003	0.00008	0.0000838	4.358E-09	0.9962	0.0004
17	0.002059	0.00295	0.0015043	5.954E-05	0.9725	0.0055
18	0.006460	0.03295	0.0345952	4.737E-04	0.6542	0.0000
19	0.000119	0.00090	0.0000312	5.675E-08	0.9986	0.0229
20	0.000001	0.00007	0.0000673	-1.022E-08	0.9981	0.0004
21	0.010329	0.00024	0.0002309	2.880E-05	0.7466	0.0000
22	0.003813	0.00177	0.0019977	1.229E-05	0.9900	0.0029
23	0.004452	0.01550	0.0012363	2.150E-04	1.0000	0.0815
24	0.000001	0.00005	0.0000088	-7.695E-09	0.9799	0.0000
25	0.020146	0.00534	0.0034918	1.602E-04	0.7438	0.0055
26	0.002641	0.00160	0.0016009	4.687E-05	0.9742	0.0029
27	0.000017	0.00002	0.0000239	4.625E-08	0.9971	0.0138
28	0.000018	0.00023	0.0002362	1.027E-07	0.9517	0.0035
29	0.023752	0.00265	0.0006713	1.491E-04	0.9890	0.0937
30	0.002641	0.00160	0.0016009	4.687E-05	0.6829	0.0000
31	0.000015	0.00004	0.0000260	-2.922E-08	0.9791	0.0035
32	0.000006	0.00016	0.0001635	3.663E-08	0.9987	0.0154

Table 6.13 Results for the thirty-two cylinders extracted

	angle [°]	delta_r [mm]	min_axis_diff [mm]	avep_ddiff [mm]	CPI	WPI
l/r	-0.0787	0.0001	0.0250	-0.0002	-0.3185	-0.4121
l/pd or r/pd	-0.1169	-0.0530	-0.0672	-0.0012	2.1901	0.0259
fa/da	0.0363	-0.0173	-0.0310	-0.0002	0.3597	0.0982
nds/pd	0.0089	-0.0461	-0.0371	-0.0004	-0.1543	-0.0378
ntol	0.0229	0.0247	0.0114	0.0007	-0.1659	-0.0477
bna/da	-0.0069	-0.0198	-0.0357	-0.0002	2.1889	0.2827
da	0.0657	0.0765	0.0569	0.0010	0.2976	0.4153

Table 6.14 Influences of design variables on the measured results for cylinders

The best extraction results were obtained for test cases 17 (small 20mm radius, 10 mm length cylinders), 6 (small 20mm radius, 80 mm length cylinders), 3 and 31 (large 200mm radius, 100 mm length cylinders) and 24 (large 200mm radius, 800 mm length cylinders). Pictures of fitted cylinders are presented in Appendix B.

6.4.2 Guidelines for Choosing Entity Extraction Input Values

The ability of choosing the correct input values improves with experience. These guidelines should be used by users that are not familiar with the process so that extreme values are not selected.

The fit accuracy must be larger or equal to the data accuracy. It does not make sense to select a fit accuracy smaller than the accuracy of the point cloud scanner machine. The fit accuracy can be twice or even three times the data accuracy if rough surfaces are to be extracted. This will make the extracting process more robust. The node size must be selected as two or three times the point pitch. The entity boundary may be found inaccurately if the node size selected is more than four times the point pitch. A node size smaller than twice the point pitch may cause refined octree nodes that do not contain points. This will stop the extraction process prematurely.

A node tolerance of 0.5 to 0.7 tends to give accurate results. The boundary of the entity will be inaccurately estimated if this value becomes smaller than 0.4. The

extraction algorithm may include nodes that have one or two points satisfying the fit accuracy but are actually not part of the entity fitted. Therefore, the found boundary may larger than the actual entity boundary. Nodes belonging to the entity may be discarded if the chosen node tolerance is larger than 0.8. These discarded nodes can be located at the boundary or in the middle of the entity.

A boundary node accuracy of about half the data accuracy tends to give accurate boundary extraction results. If this chosen value is too small (± 0.1 or less), points belonging to the entity boundary may be discarded. Values larger than 0.7 may cause points at the boundary to be included that are not part of the entity. This value is also directly proportional to the surface roughness of the entity to be fitted. If the surface roughness is high, larger values can be selected with accurate results and visa versa.

6.4.3 Comparison between Computer Times

This paragraph summarizes the computer time spent by three different computers to give the reader some insight regarding the octree creation times and entity extraction times. The three computers used are listed below.

- Intel Celeron 333 MHz with 512 MB RAM at 66 MHz
- Intel Pentium III 733 MHz with 640 MB RAM at 133 MHz
- AMD Athlon 1333 MHz with 512 MB RAM at 133 MHz

The average times for octree creation and entity extraction for the plane, sphere and cylinder test cases are listed in Table 6.15. The AMD Athlon computer outperformed the other computers as expected. The Intel Celeron computer performed much worse than the other computers as it only writes and reads to its memory at 66 MHz (half of the speed of the other two computers). As can be seen from Table 6.15 more computer time is spent to build the octree (of the planes) than time spent to extract the planes. This scenario is just the opposite in the case of extracting spheres and cylinders. The reason is the Gauss-Newton iteration steps required for sphere and cylinder fitting as their distance equations are not linear as in the case of planes.

	Celeron 333 MHz	Pentium 733 MHz	Athlon 1333 MHz
Average octree built time [s] for planes	10.3	4.5	2.3
Average plane extraction time [s]	1.3	0.7	0.2
Average octree built time [s] for spheres	14.7	5.5	2.7
Average sphere extraction time [s]	96.3	46.3	20.0
Average octree built time [s] for cylinders	64.5	25.0	12.5
Average cylinder extraction time [s]	296.7	177.8	70.5

Table 6.15 Computer times spent on building the octrees and extracting the entities

6.5 Extraction of Entities from Practical Test Cases

Figures 6.2 and 6.3 show a section of a corrugated surface measured with a Renishaw Cyclone coordinate measuring machine. The measured point pitch is 0.1 mm and the probe diameter is 0.515 mm. The points are measured as the positions of the probe centre point at any time during the scanning process. Therefore the probe radius must be added to the radius of inside curves and subtracted from the radius of outside curves to obtain the true radius of these curves.

The horizontal internal half cylinder at *A* lies at an angle of 14° to the x-axis and has a radius of 1.5 mm. Different input values for the node size, node tolerance and fit accuracy were used to try to find these values. The best results were obtained with the node size set to three times the point pitch, the fit accuracy to 50 μm (more or less the same as that of the coordinate measuring machine) and the node tolerance to 0.6. The boundary node accuracy was set to 5 μm. This value does not play a role in the final computed entity parameter values as mentioned before. The angle between the

cylinder axis and the x-axis was found to be 14.0099° and the cylinder radius as 1.483 mm. These values compare very well with the manufactured cylinder values. The final fitted cylinder is shown in Figure 6.4.

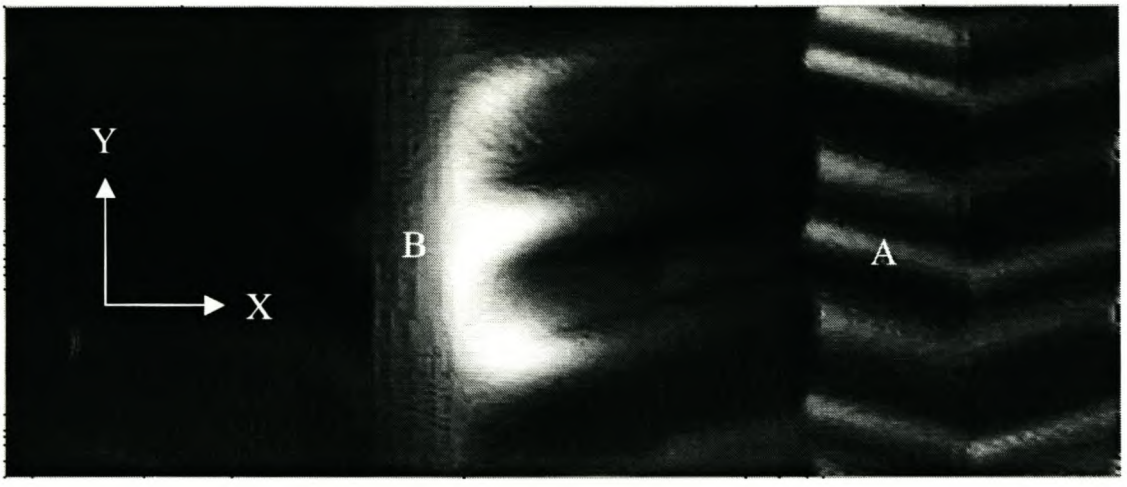

Figure 6.2 Corrugated surface plan view

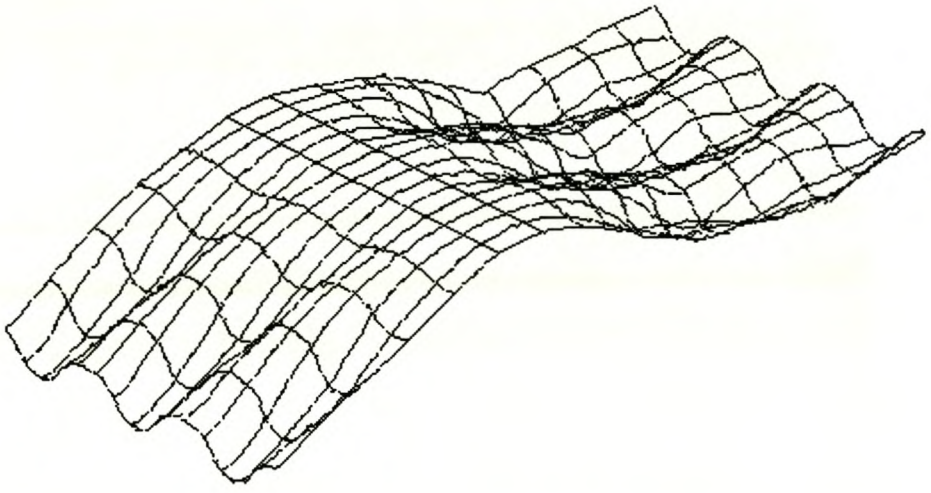

Figure 6.3 Isometric wire frame view of the corrugated surface

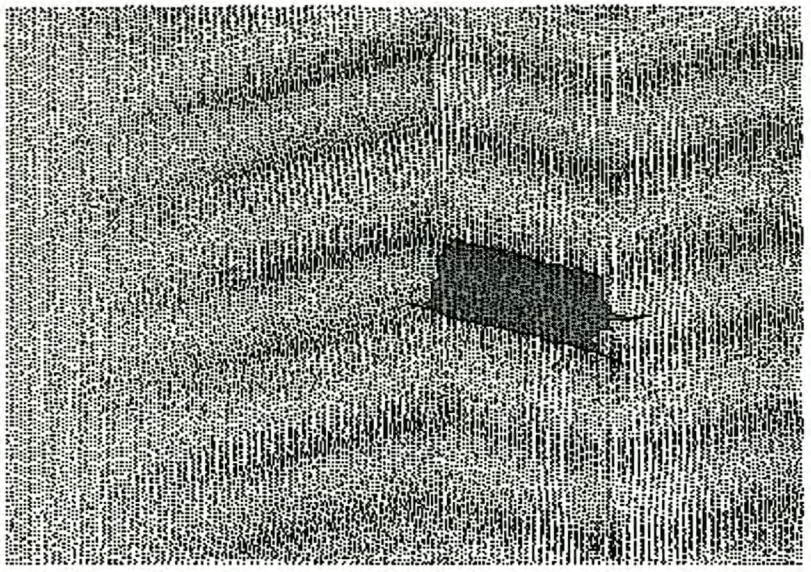

Figure 6.4 Extracted half cylinder

The horizontal open cylinder *B* (Figure 6.2) lies in the direction of the y-axis and has a radius of 17.55 mm. The best extracted cylinder results were obtained with the node size set to three times the point pitch, the fit accuracy set to 50 μm (same as the scanning accuracy of the measuring machine), the node tolerance to 0.5 and the boundary node accuracy to 5 μm.

The angle between the cylinder axis and the y-axis was found to be 0.046° and the cylinder radius as 17.5511 mm. These values compare excellently with the true parameter values. Figure 6.5 shows the final extracted cylinder. The extending fingers are data points that belong to the actual cylinder. These fingers lie at the top of the outer small cylinders with radius equal to 1.5 mm.

Figures 6.6 and 6.7 show part of the other side of this corrugated surface. The cylindrical channel lies in the direction of the x-axis and has a radius of 4 mm. The surface was scanned with a probe with a diameter of 0.999 mm and a scan point pitch of 0.5 mm was used. The best extracted cylinder results were obtained with the node size set to twice the point pitch, the fit accuracy set to 50 μm (same as the scanning accuracy of the measuring machine), the node tolerance to 0.5 and the boundary node accuracy to 5 μm.

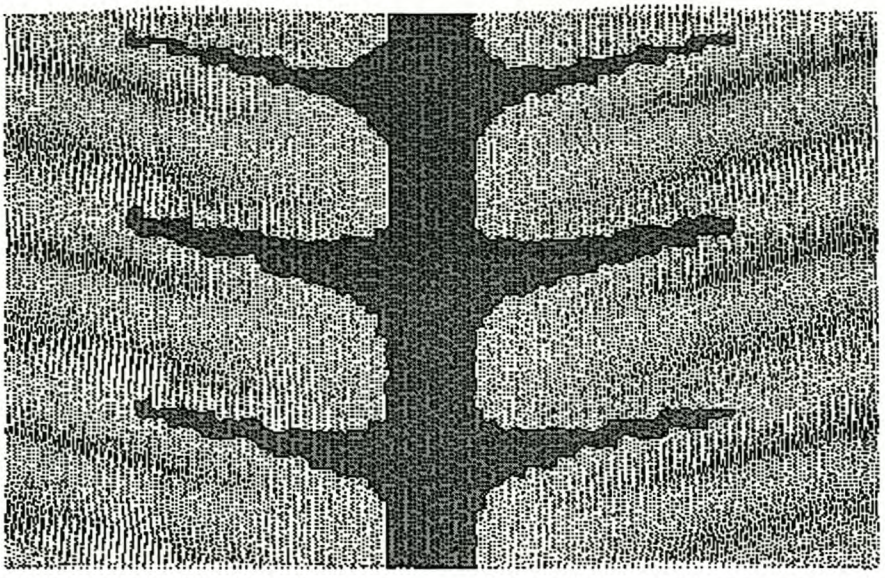

Figure 6.5 Final extracted open cylinder

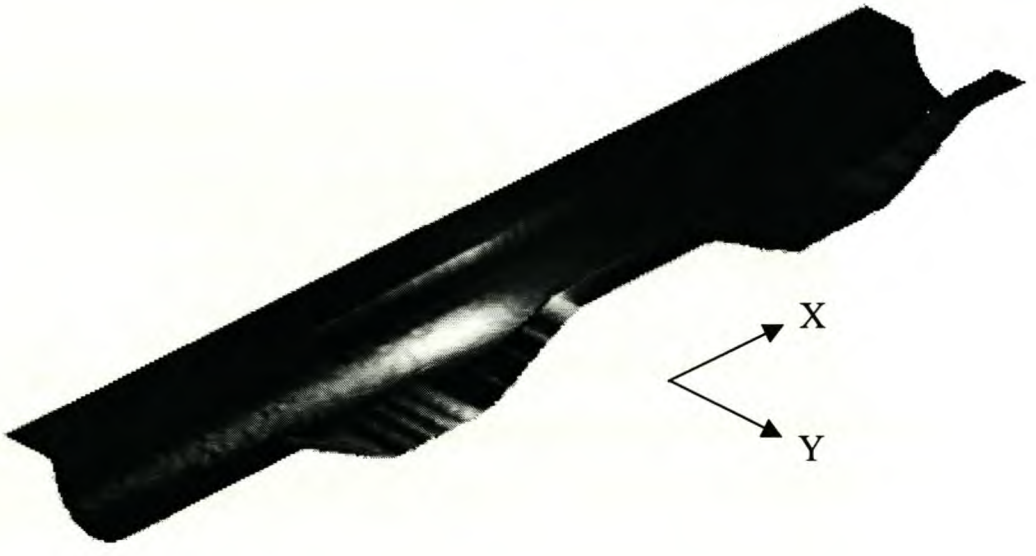

Figure 6.6 Cylindrical channel on the other side of the corrugated surface

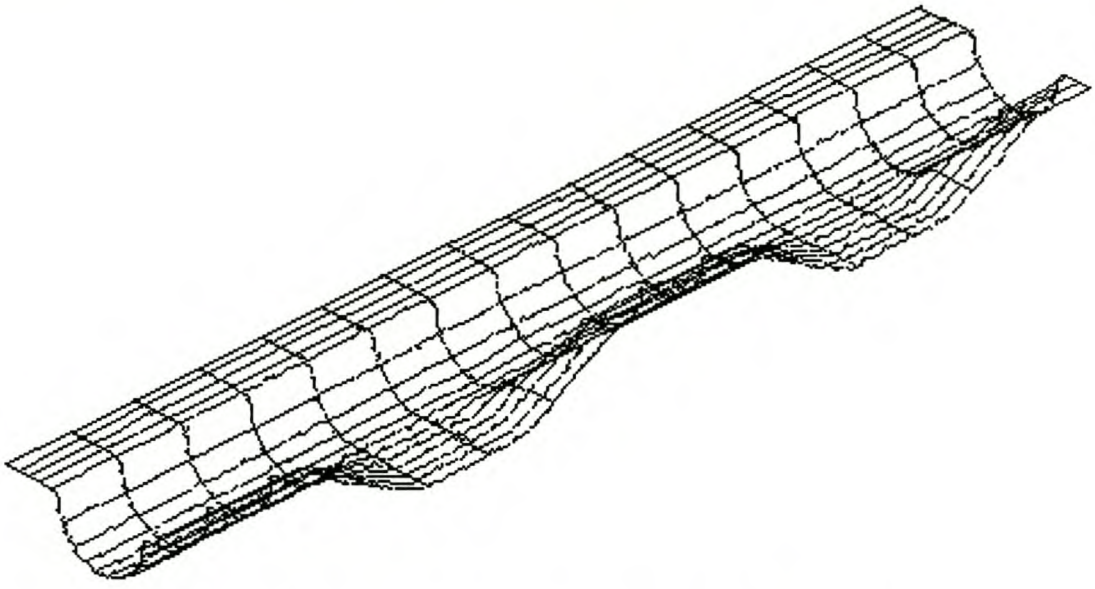

Figure 6.7 Wire frame view of the cylindrical channel

The angle between the cylinder axis and the x-axis was found to be 0.081° and the cylinder radius as 3.98144 mm after the probe radius was added for the internal cylinder. These values compare well with the true parameter values. Better values were obtained for the outside heat transfer surface as this point cloud was measured from the final aluminium cut surface. The inside of the heat transfer surface was measured from a wood model of the corrugated surface. The surface finish was not as good as that of the aluminium part. Figure 6.8 shows the final extracted cylinder. Note the cuts into the cylinder surface.

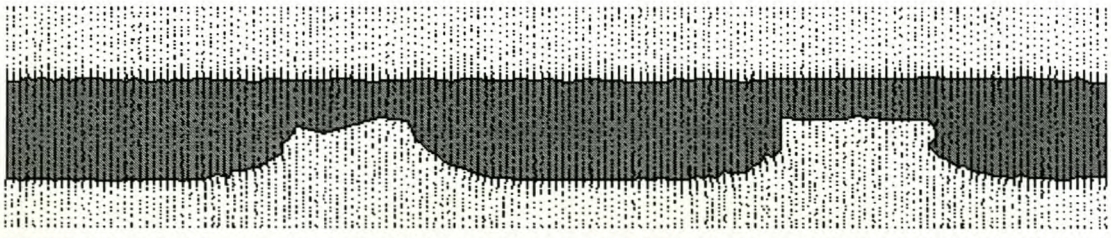

Figure 6.8 Top view of final extracted cylindrical profile

7 Conclusions

Thesis objectives were obtained from the literature review. The following objectives were met with this research.

- accurate extraction of three different types of entities (whole or only parts of planes, cylinders and spheres) from unsegmented point clouds
- accurate identification of entity boundaries
- robustness of extracting algorithm

The primitive geometric entity extraction algorithm is able to extract the three types of entities accurately from point clouds. The algorithm is able to accurately extract entities positioned close to the origin of the point cloud or far from the origin. The orientation of an entity relative to the scanned coordinate system is irrelevant for the successful extraction of the entity. The only limiting factor is that the entity physical size divided by the scanned point pitch must at least be in the order of ten. The algorithm may not extract the entity correctly if this requirement is violated. The proportions of the entity to be extracted do not matter either. Entities can be long and slender as well as short and fat in the case of cylinders for example.

The extraction algorithm finds the boundary of the entity accurately. The boundary node tolerance has the strongest influence on the accuracy to which the entity boundary will be determined. Other factors that play a role are the node size in proportion to the point pitch and the accuracy of the data points. The algorithm has the ability to find the boundary reasonably accurately where two different entities join smoothly. This type of boundary is obviously more difficult to find than a sharp edged boundary between different entities.

The algorithm performed really well as far as robustness of the extraction process is concerned. The user will be warned by the algorithm when not enough points exist to extract the entity or when initial parameters for the entity cannot be found as a result of start points that were selected too close to the entity boundary. The ability of the user to choose good values for the input parameters will improve with experience, but

the numerical experiments presented in the thesis give good guidelines for the parameter values. Obviously, the algorithm becomes more robust in the extraction process when the accuracy of the fitting process is decreased. The algorithm fulfilled the goal of extracting entities accurately and still being robust in the fitting process. It is often difficult to combine these two requirements.

The time aspect of the extracting process is only compared for the same algorithm on three different computers. The algorithm required running times of a few seconds for small entities (in the range of a 1000 points) to a few minutes for larger entities (in the range of 50000 points) on personal computers based on Pentium CPU's. Up to a minute was spent by the algorithm to construct the octree data structure for point clouds in the range of 100000 points. The times are however dependent on the processor and memory speed of a specific computer used.

The data structure used as well as the different fitting procedures is well suited for this entity extraction method. Procedures to estimate the initial entity parameters give predictions that are sufficiently accurate for the fitting process to be successful. The design of experiments case study verified the abilities of the extraction algorithm. This type of experiment tested the algorithm to some extent regarding extracting different entities with a wide range of values for the design variables.

The results shown in this work proves that this primitive geometric entity extraction algorithm can be applied with success in the reverse engineering field. It has the ability to be incorporated into other surface fitting software as an aid to the user or as an alternative to check entities that has been fitted. Therefore it can also be applied in a quality control environment as an inspection tool for primitive geometric entities.

Future work can include the following: The extraction algorithm can be combined with the work of Schreve and Basson [2000] into one unit. This combination process is aided by the use of the same octree data structure in both algorithms. The entity extraction times must be fully compared to the surface fitting times of other reverse engineering software algorithms. Both software algorithms must be used by qualified people in order to obtain reliable results. The extraction of cones and torusses can be added to the existing algorithm. Extracting different entities that have some

relationship relative to each other can also be considered. An example would be extracting a plane with a drilled hole through it where the axis of the hole has to be perpendicular to the plane surface.

8 References

Bradley, C, Vickers, GW and Milroy, M, 1994, "Reverse Engineering of Quadric Surfaces Employing Three-Dimensional Laser Scanning", Proceedings of the Institution of Mechanical Engineers, Vol. 208, Part B, Journal of Manufacture, pp. 21-28.

Chivate, P.N. and Jablokow, A.G., 1993, "Solid-Model Generation from Measured Point Data", Computer-Aided Design, Vol. 25, No. 9, pp. 587-600.

Chivate, P.N. and Jablokow, A.G., 1995, "Review of Surface Representations and Fitting for Reverse Engineering", Computer Integrated Manufacturing Systems, Vol. 8, No. 3, pp. 193-204.

Cox, MG and Jones, HM, 1989, "An Algorithm for Least Squares Circle Fitting to Data with Specified Uncertainty Ellipses", IMA Journal of Numerical Analysis, Vol. 9, pp. 285-298.

Forbes, AB, 1989, "Robust Circle and Sphere Fitting by Least Squares", National Physical Laboratory Report, DITC 153/89, November.

Forbes, AB, 1991, "Least-squares Best-fit Geometric Elements", National Physical Laboratory Report, DITC 140/89, Revised Edition, February.

Golub, GH and Van Loan, CF, 1993, **Matrix Computations**, 2nd Edition, The John Hopkins University Press, Baltimore.

Hakala, DG, Hillyard, RC, Nourse, BE and Malraison, PF, 1980, "Natural Quadrics in Mechanical Design", AUTOFACT West, Anaheim, CA (November 17-20).

Janssens, M., 1998, **A Triangular Approach to Digitising Free-form Objects for Reverse Engineering**, Ph.D. Thesis, Department of Mathematics, Katholieke Universiteit Leuven, Heverlee, Belgium.

Jones, L., 1999, Octree Source Code, http://www.marlboro.edu/~mahoney/support/alg/node41.html, (28 April 1999).

Lawson, CL and Hanson, RJ, 1995, **Solving Least Squares Problems**, Classics in Applied Mathematics 15, Society for Industrial and Applied Mathematics, Philadelphia.

Lee, YT and Fang, L, 2000, "Accurate Modelling of Complex Functional Surfaces for Mechanical Design Using Free-Form Surfaces", Transactions of the ASME, Vol. 22, pp. 236-239.

Limaiem, A, Nassef, A and El-Maraghy, HA, 1996, "Data Fitting Using Dual Kriging and Genetic Algorithms", Annals of the CIRP, Vol. 45, pp. 129-134.

Meagher, D, 1982, "Geometric Modelling Using Octree Encoding", Computer Graphics and Image Processing, Vol. 19, pp. 129-147.

Metwalli, SM, Alaa, M, Radwan, E, Abdel-Wehab, O, Shalaby, O, Moussa, YA and Hosni, YA, 1999, "Maintenance and Parts Fabrication in Reverse Engineering", ASME Design Engineering Technical Conference, Las Vegas, Nevada, September 12-15.

Mills, AF, 1992, **Heat Transfer**, Irwin, Burr Ridge, Illinois, Boston, Massachusetts, Sydney, Australia.

Milroy, MJ, Bradley, C and Vickers, GW, 1997, "Segmentation of a Wrap-Around Model Using an Active Contour", Computer-Aided Design, Vol. 29, No. 4, pp. 299-320.

Milroy, MJ, Weir, DJ, Bradley, C and Vickers, GW, 1996, "Reverse Engineering Employing a 3D Laser Scanner: A Case Study", International Journal of Advanced Manufacturing Technology, Vol. 12, pp. 111-121.

Nassef, AO, Ashraf, AM and Metwalli, SM, 1999, "Accuracy and Fitting-time Minimization in the Reverse Engineering of Prismatic Features", ASME Design Engineering Technical Conference, Las Vegas, Nevada, September 12-15.

O'Connor, P.D.T., 1991, **Practical Reliability Engineering**, Third Edition, John Wiley & Sons, Chichester.

Park, S and Jun, Y, 2000, "A Face-Based Reverse Engineering Approach to Primitive Analytic Surfaces", International CIRP Design Seminar, May 16-18, Haifa, Israel.

Perucchio, R, Saxena, M and Kela, A, 1989, "Automatic Mesh Generation from Solid Models Based on Recursive Spatial Decompositions", International Journal for Numerical Methods in Engineering, Vol. 28, pp. 2469-2501.

Piegl, L and Tiller, W, 1997, **The NURBS Book**, 2nd Edition, Springer-Verlag, Berlin.

Press, W.H., Teukolsky, S.A., Vetterling, W.T. and Flannery, B.R., 1997, **Numerical Recipes in C**, Second Edition, Cambridge University Press, Cambridge.

Ramakrishnan, CV, Ramakrishnan, S, Kumar, A and Bhattacharya, M, 1992, "An Integrated Approach for Automated Generation of Two/Three Dimensional Finite Element Grids Using Spatial Occupancy Enumeration and Delaunay Triangulation", International Journal for Numerical Methods in Engineering, Vol. 34, pp. 1035-1050.

Rao, SS, 1996, **Engineering Optimisation Theory and Practice**, 3rd Edition, John Wiley & Sons Inc., New York.

Sapidis, NS and Besl, PJ, 1995, "Direct Construction of Polynomial Surfaces from Dense Range Images through Region Growing", ACM Transactions on Graphics, Vol. 14, No. 2, pp. 171-200.

Samuel, GL and Shunmugam, MS, 1999, "Evaluation of Straightness and Flatness Error Using Computational Geometric Techniques", Computer Aided Design, Vol. 31, pp. 829-843.

Schreve, K and Basson, AH, 2000, "Edge Detection in Reverse Engineering Using a Virtual CMM", ASME 2000 Design Engineering Technical Conferences, September 10-13, Baltimore, Maryland, USA.

Shephard, MS and Georges, MK, 1991, "Automatic Three-Dimensional Mesh Generation by the Finite Octree Technique", International Journal for Numerical Methods in Engineering, Vol. 32, pp. 709-749.

Taubin, G, 1991, "Estimation of Planar Curves, Surfaces, and Nonplanar Space Curves Defined by Implicit Equations with Applications to Edge and Range Image Segmentation", IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol. 13, No. 11, pp. 1115 - 1138.

Thompson, W.B., Owen, J.C., de St. Germain, H.J., Stark (Jr), S.R. and Henderson, 1999, "Feature-Based Reverse Engineering of Mechanical Parts", IEEE Transactions on Robotics and Automation, Vol. 15, No. 1, pp. 57-66.

Van Vliet, WP and Schellekens, PH, 1996, "Accuracy Limitations of Fast Mechanical Probing", CIRP Annals, Vol. 45, pp. 483-487.

Várady, T, Martin, RR and Cox, J, 1997, "Reverse Engineering of Geometric Models – an Introduction", Computer Aided Design, Vol. 29, No. 4, pp. 255-268.

Vörös, J, 2000, "A Strategy for Repetitive Neighbour Finding in Octree Representations", Image and Vision Computing, Vol. 18, pp.1085-1091.

Werghi, N, Fisher, R, Robertson, C and Ashbrook, A, 1999, "Object Reconstruction by Incorporating Geometric Constraints in Reverse Engineering", Computer Aided Design, Vol. 31, pp. 363-399.

Yu, Y, Wu, M and Zhou, J, 1996, "An Octree Algorithm for Dynamic Interference Detection", ASME 1996 Design Engineering Technical Conferences, August 18-22, Irvine, California.

A Appendix A Algorithm Descriptions

A.1 Entity Fit Procedures

This section explains how the three different entities are fitted as discussed in Chapter 3. Flow charts are used for this purpose.

A.1.1 Planes

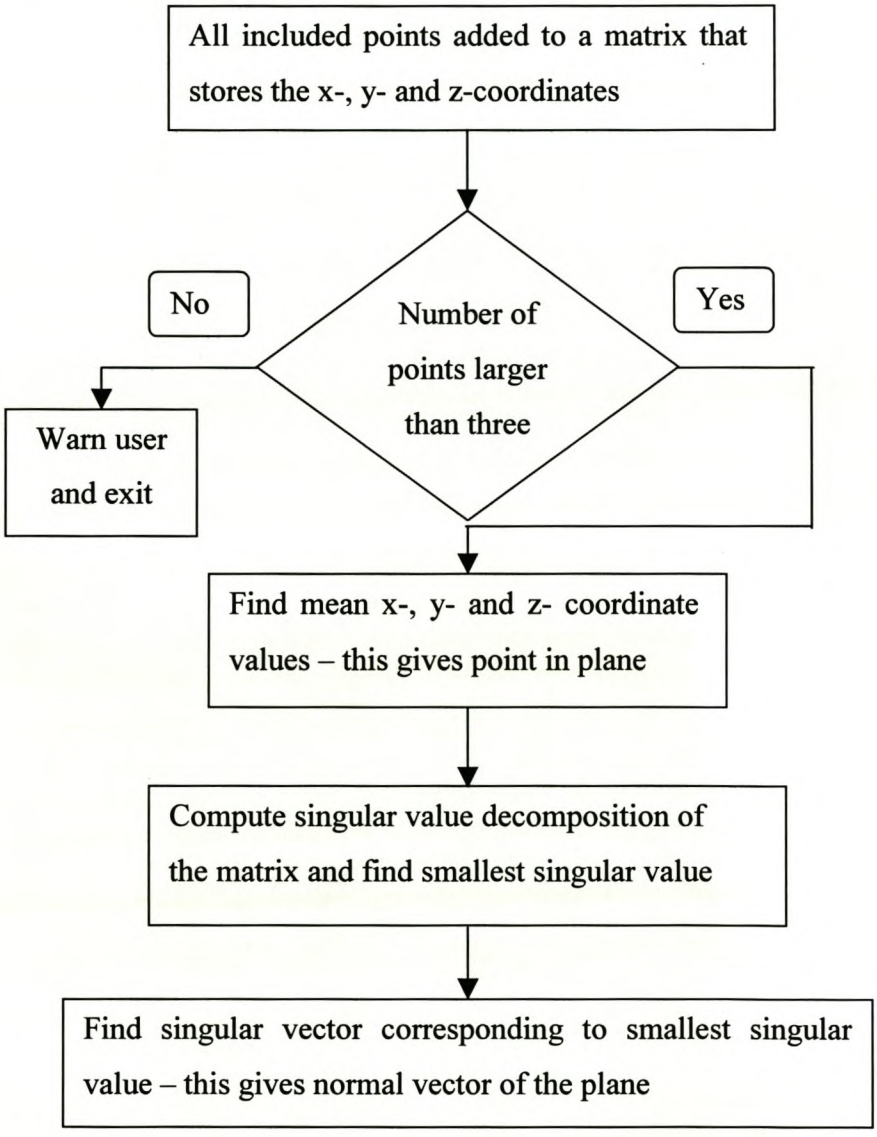

Figure A.1 Flow chart of plane fitting process

A.1.2 Spheres

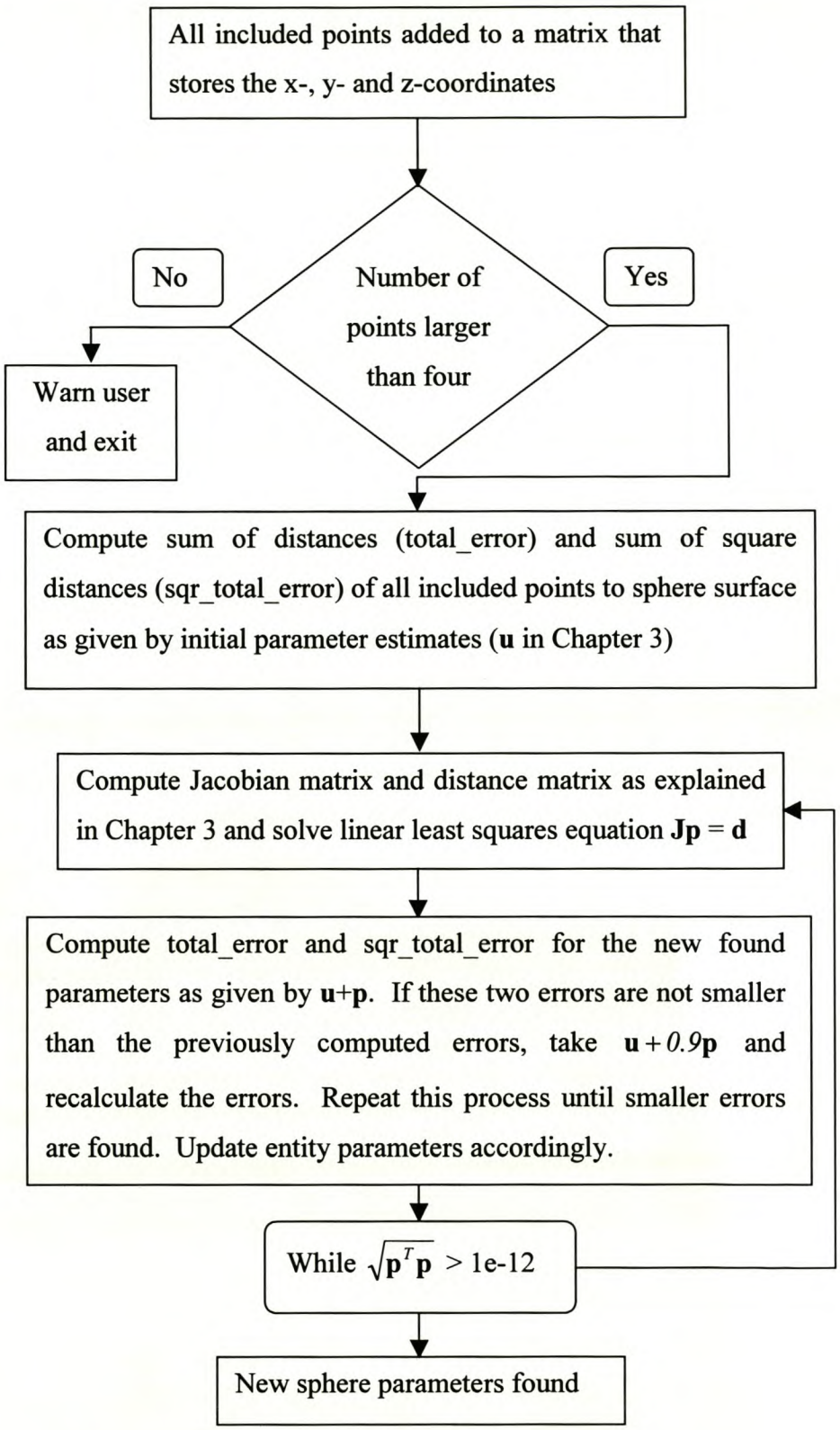

Figure A.2 Flow chart of sphere fitting process

A.1.3 Cylinders

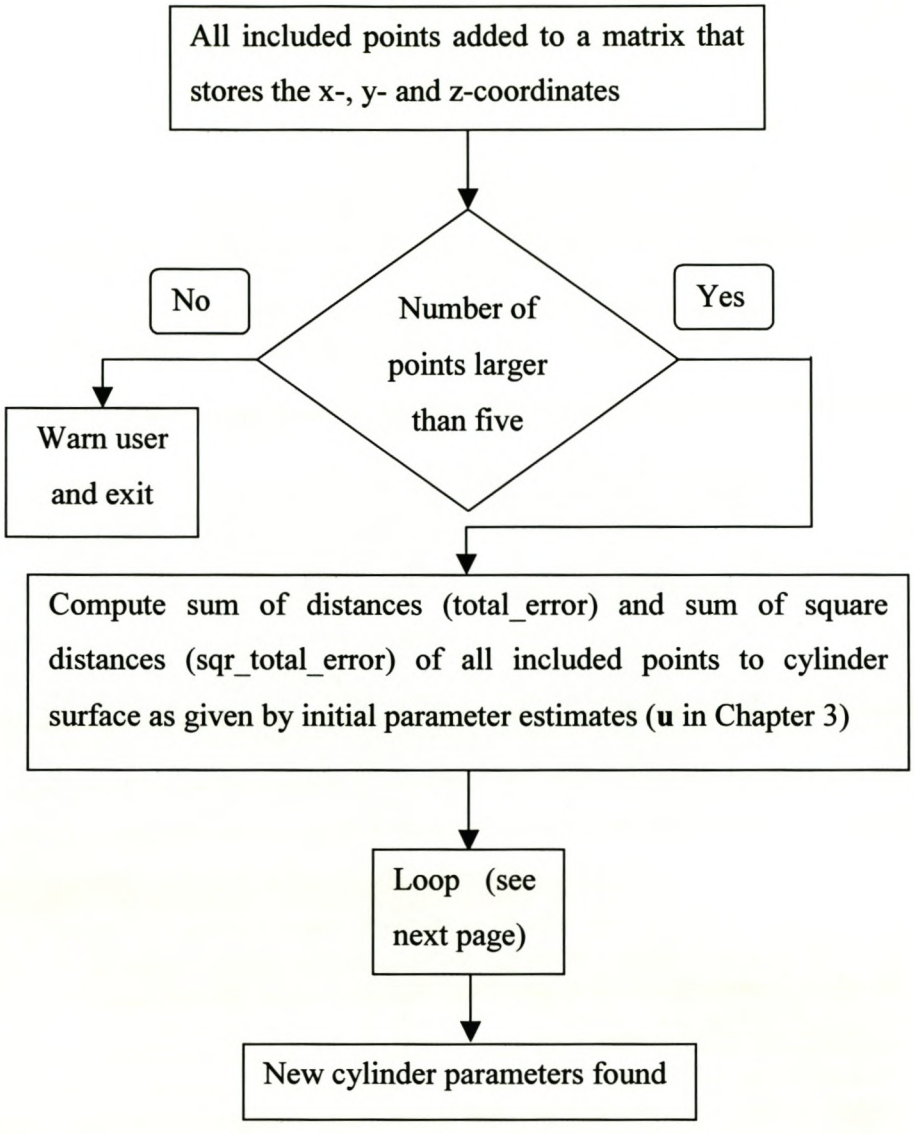

Figure A.3 Flow chart of cylinder fitting process

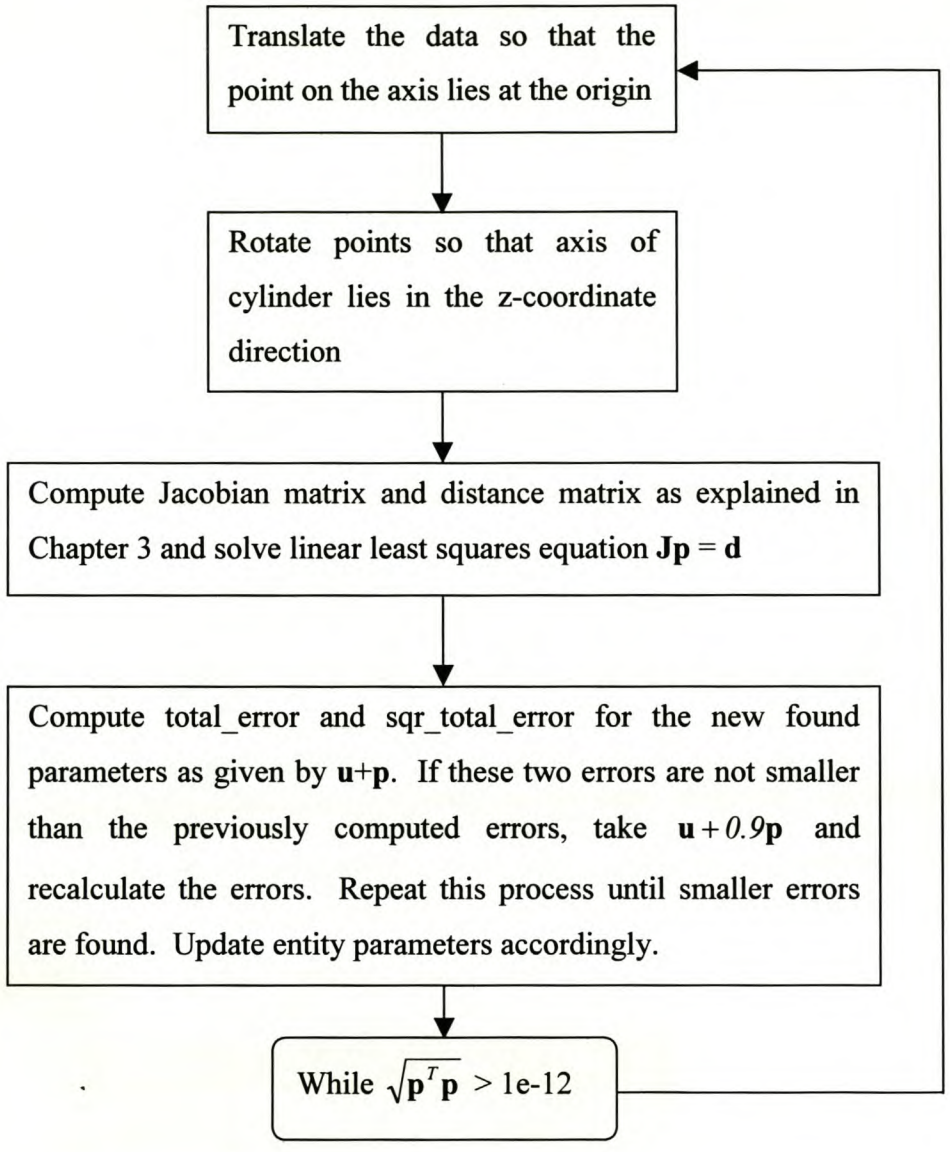

Figure A.4 Flow chart of cylinder fitting process loop of Figure A.3

A.2 The Octree Data Structure

The octree implementation uses an object orientated programming approach adapted from Jones [1999]. Object orientated programming allows the programmer to build data and procedures to manipulate the data into the same structure. An octree-node object and an octree object were developed for creating the octree data structure. The octree-node object contains data about the node itself. It has no procedures, but has storage for the following data used to build the octree (using C++ syntax):

- OctreeNode *Parent
- OctreeNode **children
- double Cx, Cy, Cz
- int first_datapoint
- int np

**Parent* is a pointer to the parent of a specific node. ***children* is a pointer to the eight child nodes of a specific node if it exists. *Cx*, *Cy* and *Cz* are the minimum x, y and z values of the coordinates that specifies a certain node as shown in Figure A.5.

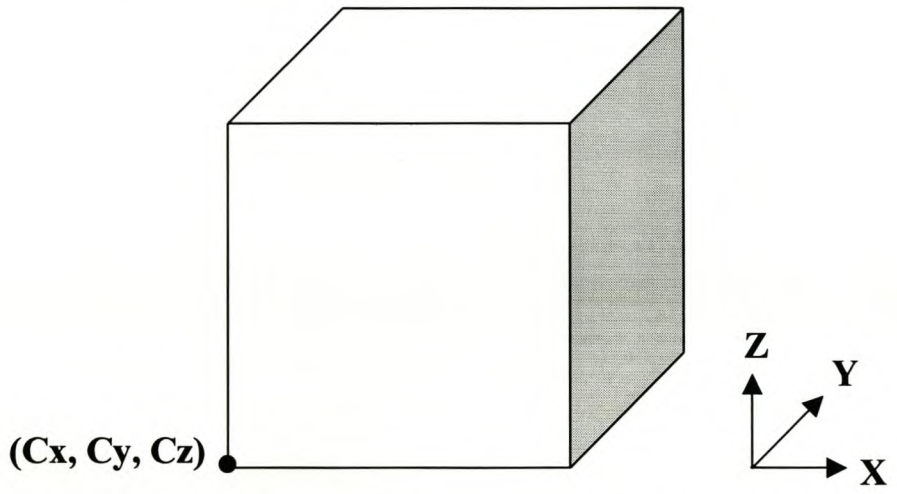

Figure A.5 Bottom left front corner of a node

For leaf nodes *first_datapoint* points to the first data point (in the data list containing all the point coordinates) that belongs to this node. The number of points in a node is given by *np*.

The octree object has the following variables and procedures used to build the octree (using C++ syntax):

- double Lx0, Ly0, Lz0
- double ppitch
- cPoint *PointArray
- int n
- double X0, Y0, Z0

- Octree(const char*filename, double pointpitch, double multfactor, OctreeNode *&pNode, bool UseMinLength, bool CubicBoxes, int MinPoints)
- void FindPointInOctree(OctreeNode *pNode, cPoint aPoint, int levels)
- bool CheckLine(char aLine[80])
- double dMin(double x, double y)
- double dMax(double x, double y)
- double GetAxisVal(int index, int axis)
- int GetFirstPoint(APOINT pNodeMin, APOINT pNodeMax, int N, int LN)
- void GetRootSize(bool CubicBoxes)
- int MidP(double Val, int col, int L, int LN)
- void ReadData(const char *filename)
- void SortDA(int N, int LN, int col)
- void Swap(int L, int U)
- OctreeNode *BuildTree(double X, double Y, double Z, int N, int LN, int level, OctreeNode *PreviousNode, int NodeNum, double multfactor, bool UseMinLength, int MinPoints)

Figure A.6 displays a flow chart of the procedure to build the octree data structure. The procedure requires seven input variables. The variable filename is the name of the text file that contains the data points. Pointpitch is the distance in millimetre between consecutive points measured by the point-scanning machine. The minimum size of the octree nodes is calculated from the point pitch multiplied with the multiplication factor. The root node is created outside the procedure and its memory address is send to the procedure. CubicBoxes is a boolean value and if it is true, the octree nodes will be cubical in shape. If CubicBoxes boxes is set to false the nodes may have side lengths that are not equal in length. UseMinLength is a boolean value and if it is false, the division of the nodes will continue until the maximum side length is equal to the point pitch multiplied with the multiplication factor, otherwise the minimum side length will be used as the stop criterium. MinPoints is an integer value that determines how many points a node must have to stop the division process. If

this value is set to zero the division process will continue until the nodes contain no points.

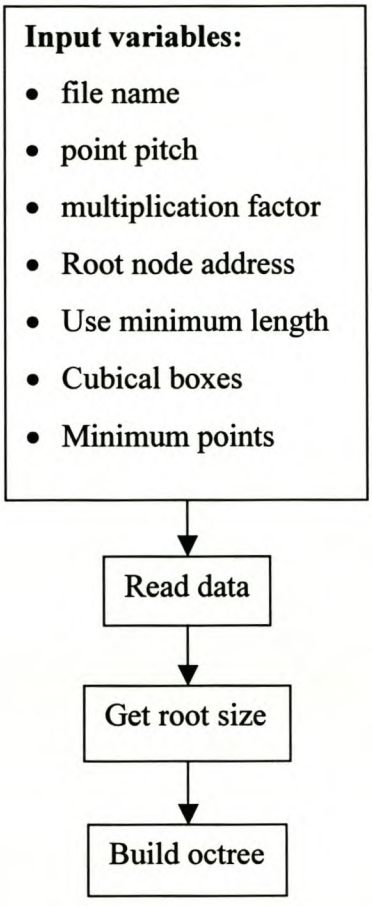

Figure A.6 Flow chart of octree data structure creation

The procedure ReadData reads the data from the file specified by the variable filename. Each line is checked to verify that it contains the three coordinates of a point in the point cloud. If it does not, that line is ignored. The point data is stored in a list containing a point class for each point. The point class contains variables for the x-, y- and z-coordinate of a point as well as procedures to manipulate the point data.

GetRootSize is a procedure that determines the size of the root node. It searches through the point data and finds the minimum and maximum x-, y- and z-coordinate value. The node size is made slightly bigger so that all the points will lie inside the

root node. The initial root node is a cube determined by the maximum x, y or z distance if CubicBoxes is set to true.

The recursive build tree procedure is further described in the flow chart of Figure A.7. Input parameters for this procedure include (using C++ syntax):

- OctreeNode *pNode is a pointer to the node to be divided
- the coordinates of the corner of the node, double X, double Y and double Z
- int N is the first index to a point in the data array that belongs to this node
- int LN is the last index to a point in the data array that belongs to this node
- int level is the current level of division for pNode

The procedure first determines what the size of the parent node's children will be. A new leaf node will be created if this size is smaller or equal to the value of the minimum size specified by the user or if the parent satisfies the minimum amount of points specification. Eight new children will be created for the parent node if none of these criteria are satisfied. The process is repeated for all the children until the stop criteria are met. The procedure continues with the refinement of the next child node whenever a child node becomes a leaf node. This is how the tree structure is created.

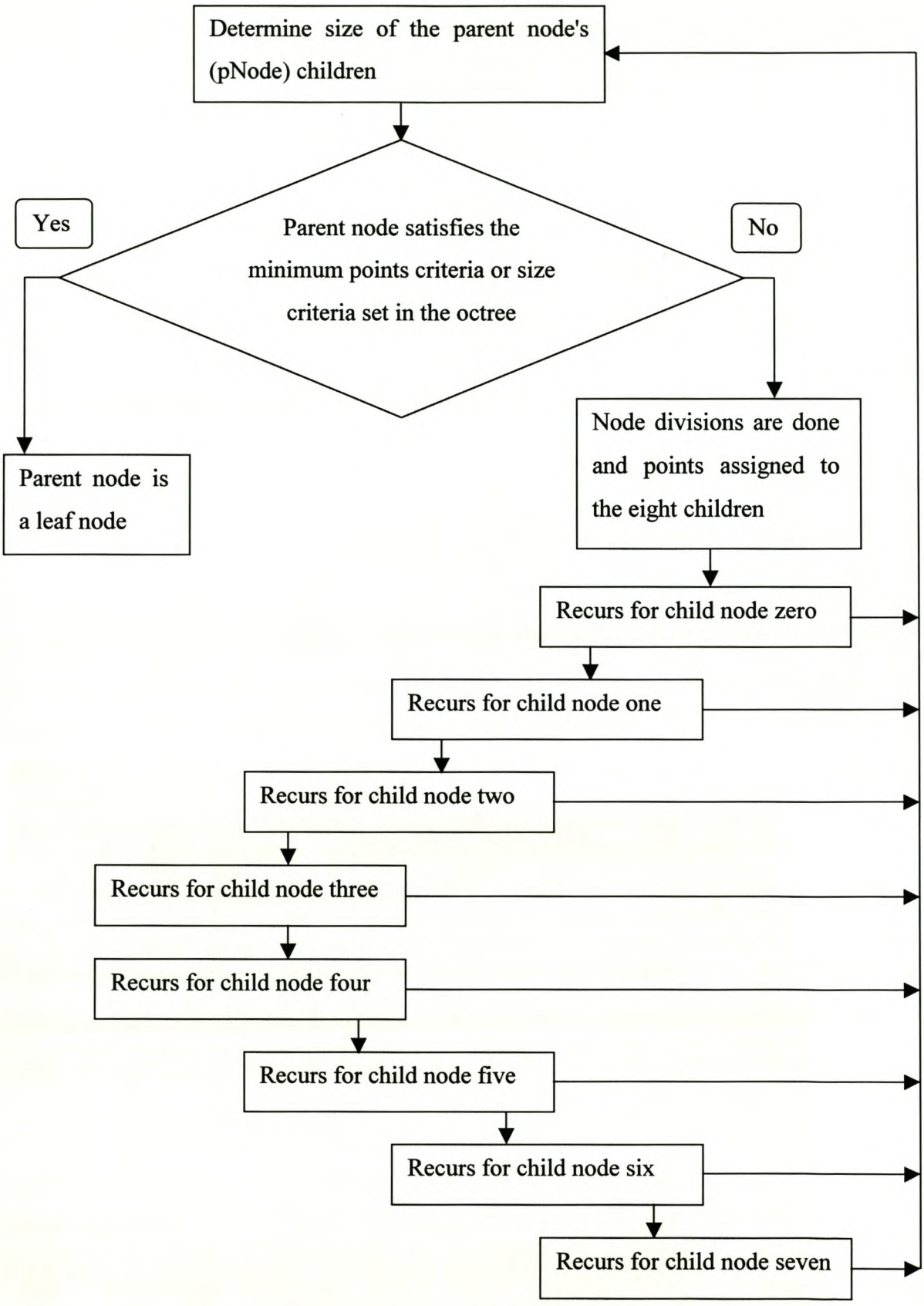

Figure A.7 Flow chart of build tree procedure

A.3 The Neighbour Search Algorithm

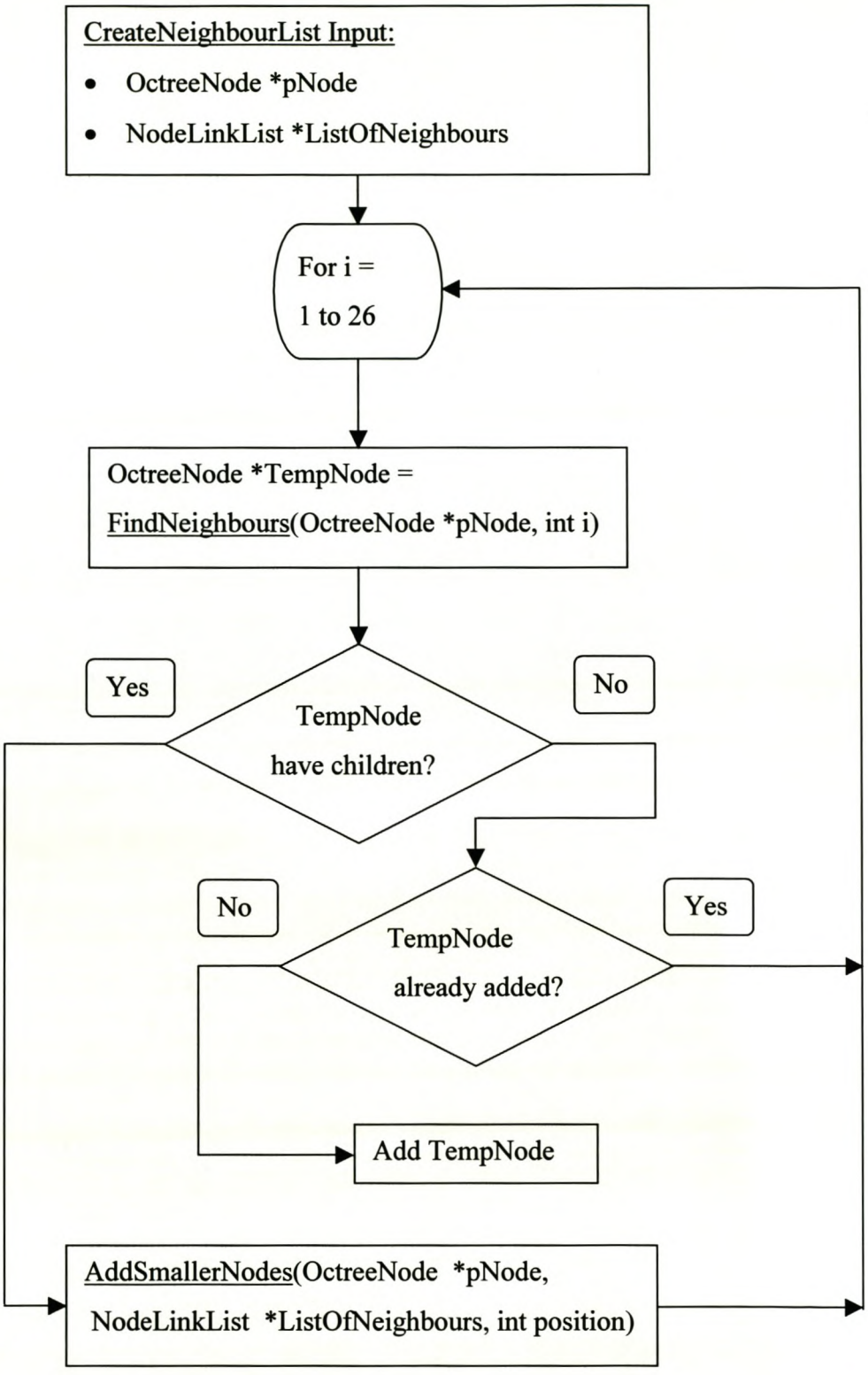

Figure A.8 Create neighbour list algorithm

The neighbour search algorithm finds all the neighbour nodes of the node pNode and adds these nodes to the list ListOfNeighbours. The algorithms FindNeighbours and AddSmallerNodes are explained later in this section. Each *i* corresponds to a certain neighbour node as described in Figures 4.5 - 4.7. FindNeighbours find the neighbour node (TempNode) that corresponds to the position *i*. AddSmallerNodes is called if TempNode has children. If this is not the case TempNode is added to the list ListOfNeighbours if it is not already added. This process is repeated for all twenty-six neighbours.

FindNeighbours adds or subtracts binary digits to the binary position numbers (x, y and z) of the original node (pNode). The algorithm calculates the neighbour node position numbers from the relative position of each of the twenty-six neighbours. It returns the neighbour node by searching up the octree for the common parent and then by searching down the octree until the neighbour with the calculated binary position numbers is found. It returns the original node (pNode) if this node lies on the boundary of the root node.

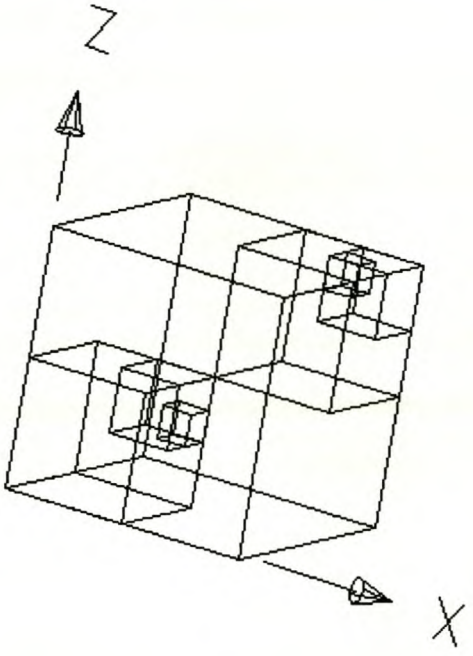

Figure A.9 Root node with one of the leaf nodes located in the root node top side

Figure A.9 shows the octree root node with a top right leaf node (level three) located in the top side of the root node. This leaf node has binary position numbers of x:110, y:110 and z:111. A top neighbour for this leaf node does not exist. The algorithm will try to add one binary digit to the z-position binary number because the neighbour will lie in the positive z-direction. This original node's binary number consists of three 1's and therefore the algorithm will realise that the node lies in the top boundary of the root node. It will therefore return the original node (pNode). This node will not be appended in the neighbour node list because it has already been added. The same scenario happens at all the boundary sides of the root node. Either the x-, y- or z-position binary numbers of the original node will all be zero's or one's and therefore the algorithm will realise that the original node lies on the boundary of the root node.

The algorithm AddSmallerNodes works on the basis that the found neighbour node is sent to the algorithm because it has children. The neighbour node's position relative to that of the original node is known. Refer back to Figures 4.5 – 4.7 for neighbour node positions information. If the neighbour node differs in three axis directions, only one of its children can be a true neighbour of the original node as shown in Figure A.10.

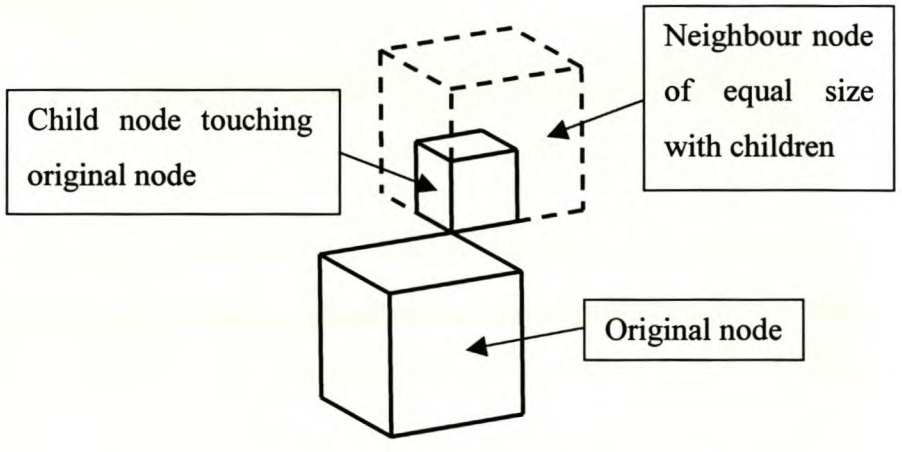

Figure A.10 Neighbour node that differs in three axis directions from the original node

Two of the equal sized neighbour node's children can be neighbour nodes of the original node if the neighbour node differs in two axis directions as shown in Figure A.11. Figure A.12 shows the last case where four of the neighbour node's children can be actual neighbour nodes of the original node, if the equal sized neighbour (not a leaf node) differs in only one axis direction.

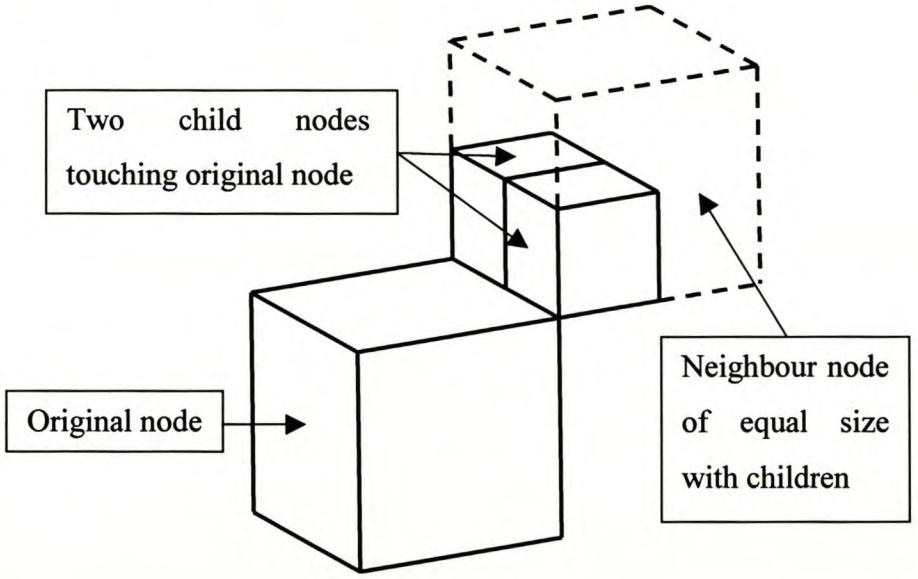

Figure A.11 Neighbour node that differs in two axis directions from the original node

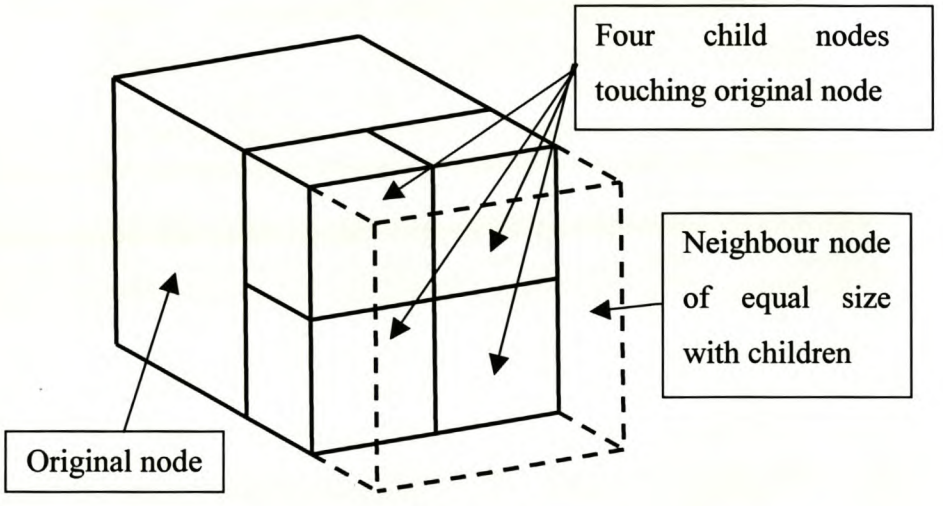

Figure A.12 Neighbour that differs in only one axis direction from the original node

AddSmallerNodes checks if the child nodes have children as well and perform the same procedure to identify their children that are neighbours of the original node. This algorithm is recursive and searches the neighbour node until no more children are found.

A.4 The Grow Entity Algorithm

The following flow chart explains the basic steps of the algorithm.

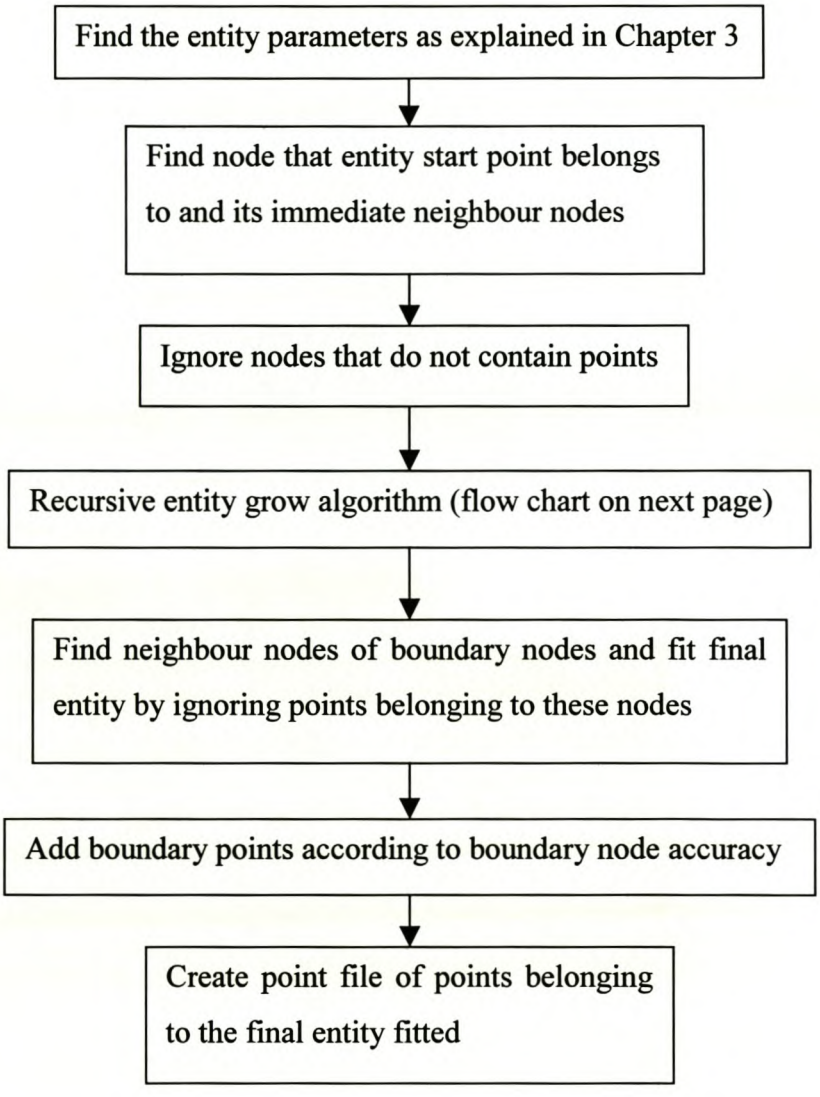

Figure A.13 Flow chart explaining the entity grow algorithm

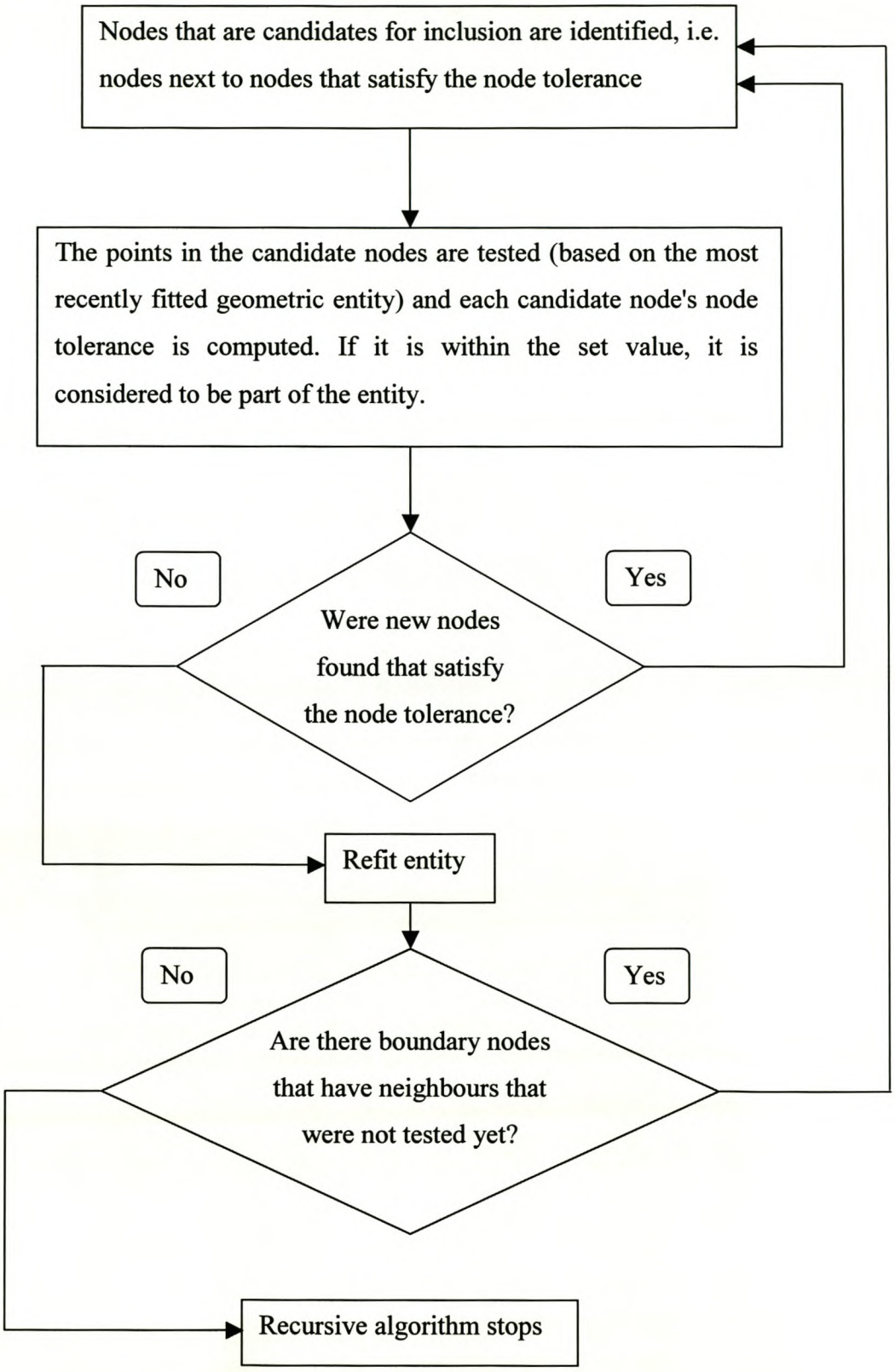

Figure A.14 Recursive entity grow algorithm

A.5 AutoCAD User Interface

During the development of the entity extraction algorithm the need was realized for a visualization method. The general modelling tool AutoCAD 2000 together with VBA (Visual Basic interpreter inside AutoCAD) fulfilled this purpose. This AutoCAD user interface is explained in this paragraph.

The user starts the entity extraction algorithm and selects a text file containing the point cloud. The octree is created with the average pitch size and node size factor provided by the user. An intermediate file is then created. The data in this file are then displayed in AutoCAD as a point cloud by using the AutoCAD interface. Start point(s) is/are then selected by the user from this point cloud. The entity extraction algorithm compute initial values for the entity parameters (in the case of spheres and cylinders) from the selected start points.

The user continues by providing the error tolerance, node tolerance and boundary error tolerance. The entity is then fitted to these values. A point file that includes all the points that belong to the entity is then created. The entity points can then be displayed in AutoCAD in a user-selected layer on top of the original point cloud.

The user has the opportunity to visualize the influence that his/her selected input values had on the entity extraction. All entities drawn can be deleted easily with the AutoCAD program. The user can fit the same entity with different input values and draw this entity in a different layer. Each time the entity is refitted a text file is created with the determined entity parameters. The user can obtain the best extracted entity by continuing with this process. All of the pictures in Appendix B were created with this program with only the final entity fitted shown.

B Appendix B Test Cases

This appendix is included in the thesis for completeness of the design of experiments case study. It provides pictures of the different entities fitted in the case study. An entity picture is given for each of the cases listed in the tables of Chapter 6. It is thus possible for the reader to interpret the different test cases better. The solid lines in the pictures are the boundaries of the entities. Some comments will also be provided throughout to explain different scenarios and entity extraction effects. Different sizes of entities are grouped together.

B.1 Planes

This paragraph shows the extracted small planes (20mm x 20mm) and extracted large planes (200mm x 200mm) of Chapter 6. Planes showing the same extraction properties are grouped together.

B.1.1 Small Planes

The seven planes (1, 5, 7, 9, 11, 13 and 15) show the same extraction effect where most of the inner points are included but most of the boundary nodes' points are not included. The reason is that the boundary node accuracy value was set too accurate. Therefore some points belonging to the plane were discarded. The node size determined the size of the indents from the boundaries where points were discarded. Planes 1, 9 and 11 had an octree node size of three times the point pitch, while planes 5, 7, 13 and 15 had a node size of five times the point pitch.

The seven planes (17, 19, 21, 23, 27, 29 and 31) show the same extraction effect where most of the inner points are included but points outside the plane boundary are also included. Planes 17, 23, 27 and 29 show the effect more clearly as their data accuracy (0.05 mm) was not as tight as the data accuracy (0.01mm) of planes 19, 21 and 31.

Planes 3 and 25 were the best extracted planes. Plane 25 had very tight values set to the fit accuracy and node tolerance. The point data was also very accurate (0.01mm volumetric error in data points). Plane 3 had less tight values but the boundary node accuracy was very tight (one tenth of the data accuracy) with a fit accuracy of three times the data accuracy.

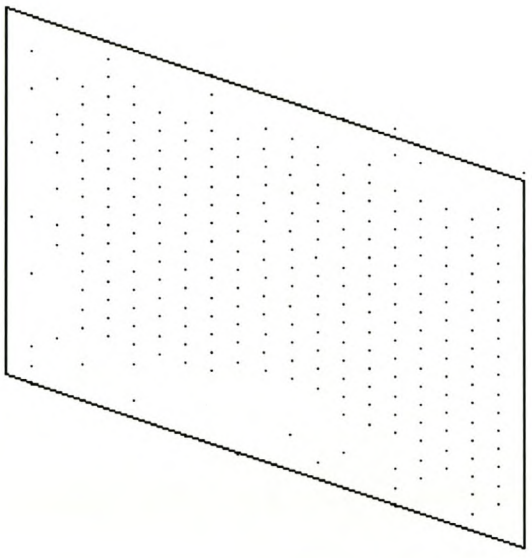

Figure B.1 Extracted plane number 1

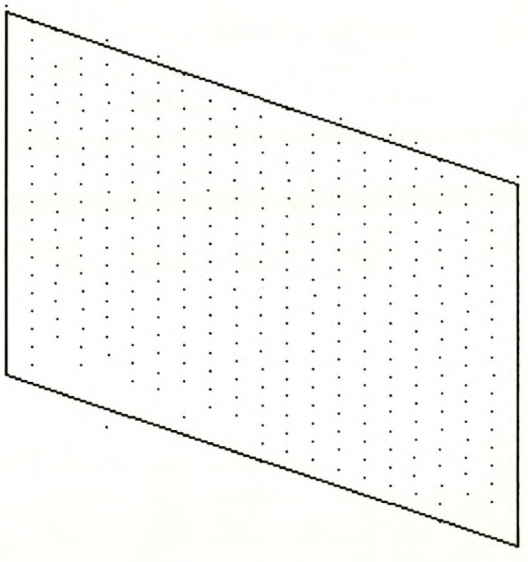

Figure B.2 Extracted plane number 3

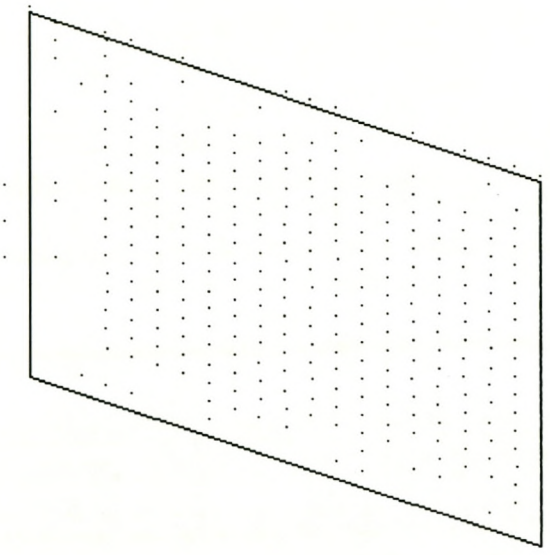

Figure B.3 Extracted plane number 5

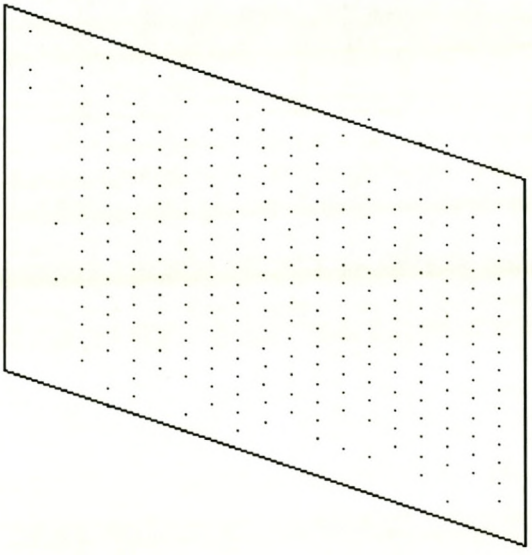

Figure B.4 Extracted plane number 7

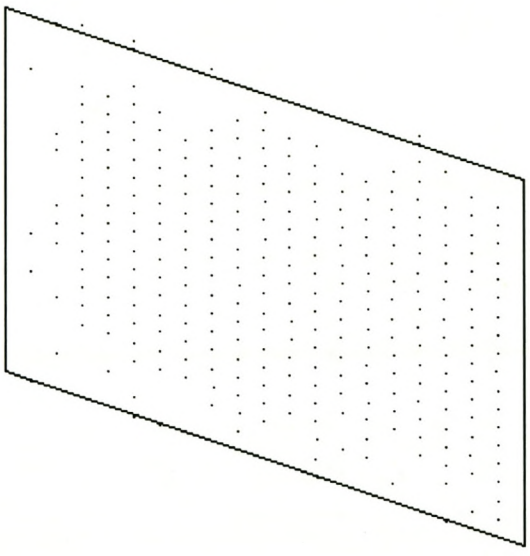

Figure B.5 Extracted plane number 9

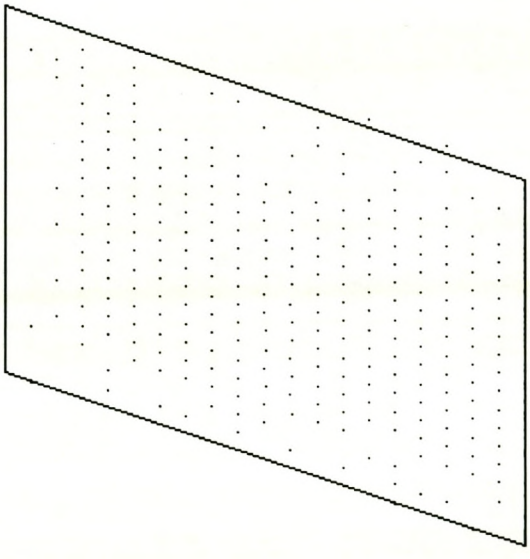

Figure B.6 Extracted plane number 11

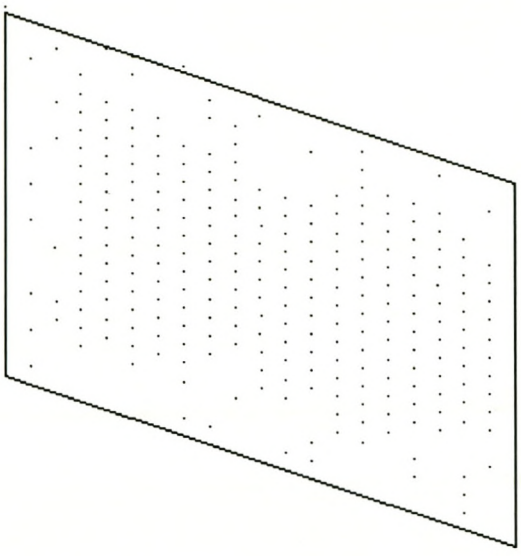

Figure B.7 Extracted plane number 13

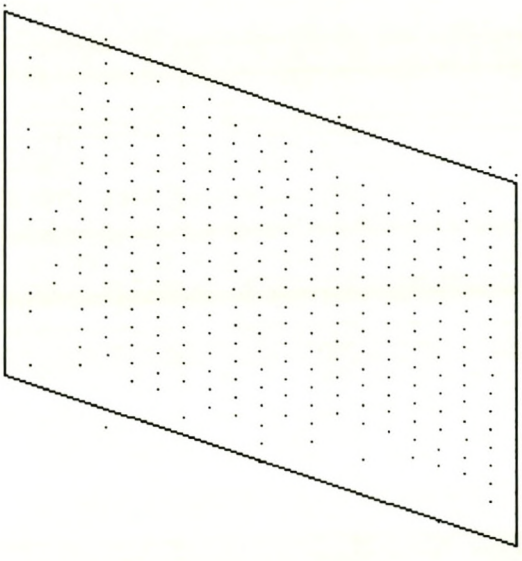

Figure B.8 Extracted plane number 15

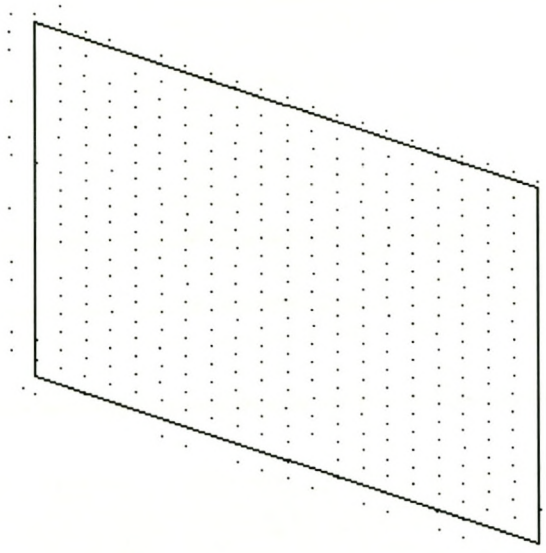

Figure B.9 Extracted plane number 17

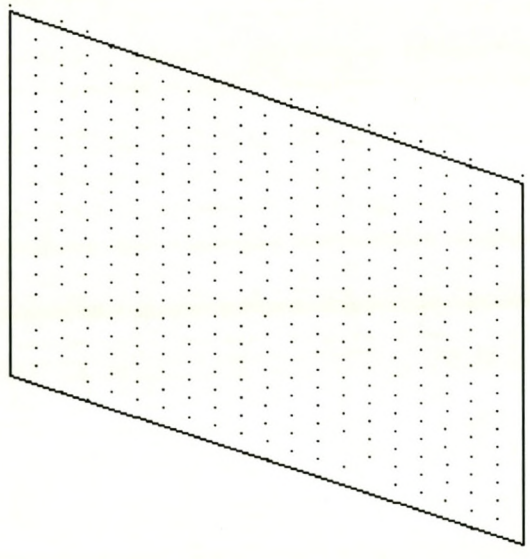

Figure B.10 Extracted plane number 19

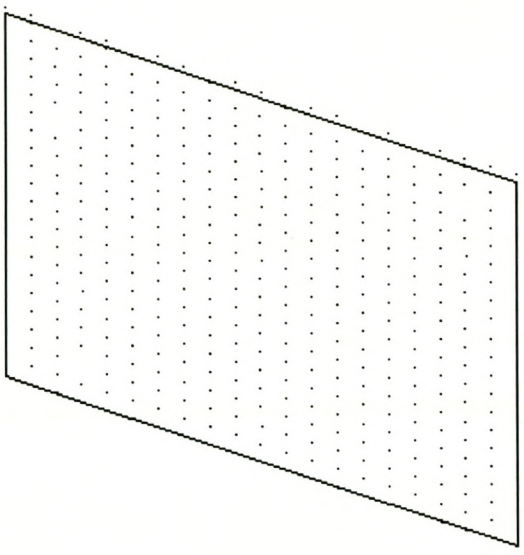

Figure B.11 Extracted plane number 21

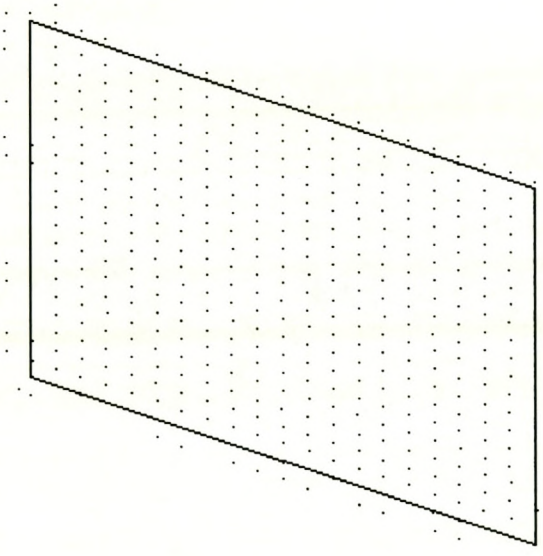

Figure B.12 Extracted plane number 23

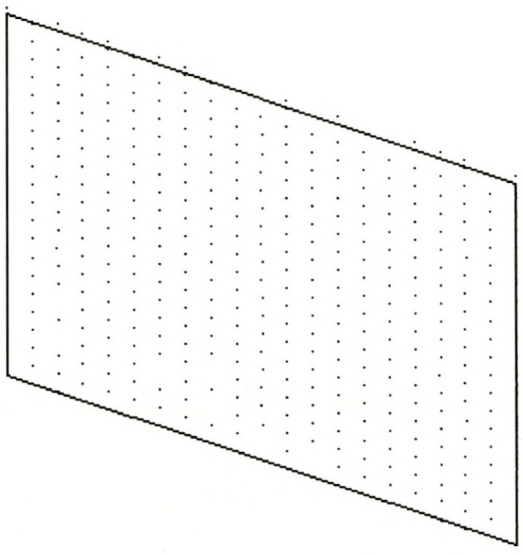

Figure B.13 Extracted plane number 25

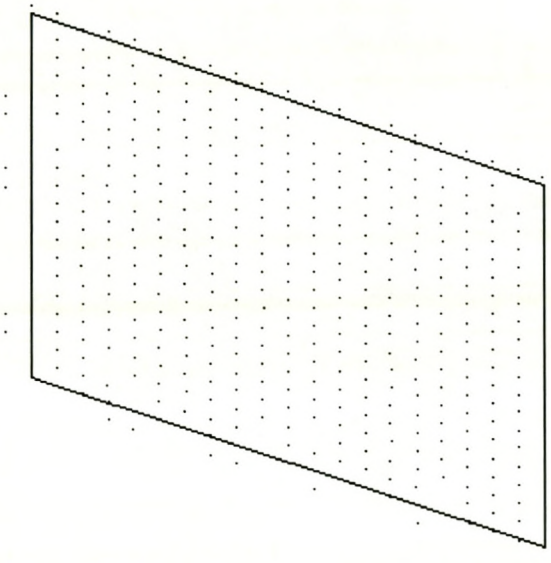

Figure B.14 Extracted plane number 27

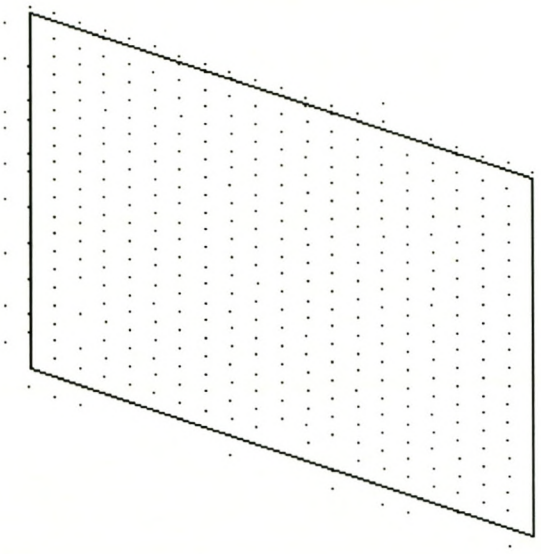

Figure B.15 Extracted plane number 29

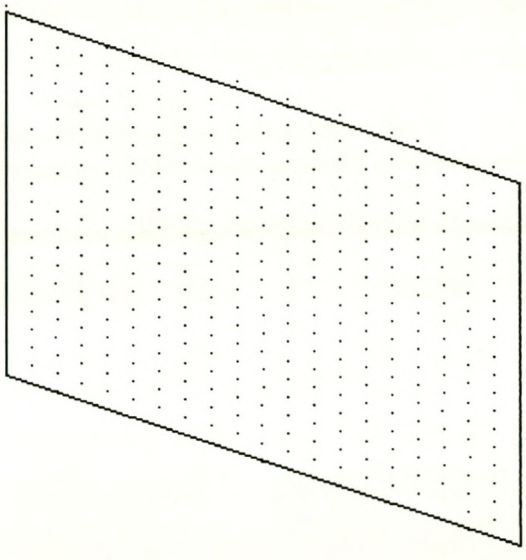

Figure B.16 Extracted plane number 31

B.1.2 Large Planes

The eight planes (2, 4, 6, 8, 10, 12, 14, 16) show the same extraction effect where most of the inner points are included but most of the boundary nodes' points are not included. Planes 2, 4, 10 and 12 shows this effect to a lesser extend as their node size was only three times the average point pitch compared to five times for planes 6, 8, 14 and 16. The same reasons (given for the small planes which had this result) apply to these eight planes.

The six planes (22, 24, 26, 28, 30, 32) have some points included that fall outside the boundary of the plane or some points belonging to the entity boundary that were discarded. The reason is that the boundary node accuracy was set to half the data accuracy, which is five times better than that of the eight previously mentioned (paragraph above) large planes. Planes 22, 26 and 32 have some points included outside the boundary of the plane because the fit accuracy (0.05mm) was not as strict as the fit accuracy (0.01mm) of planes 24, 28 and 30. These last three planes have some points belonging to the boundary that were discarded.

Planes 18 and 20 were the best extracted planes from the large planes. Both had a node size of three times the average point pitch, a node tolerance of 0.5 and a boundary node accuracy equal to half the fit accuracy.

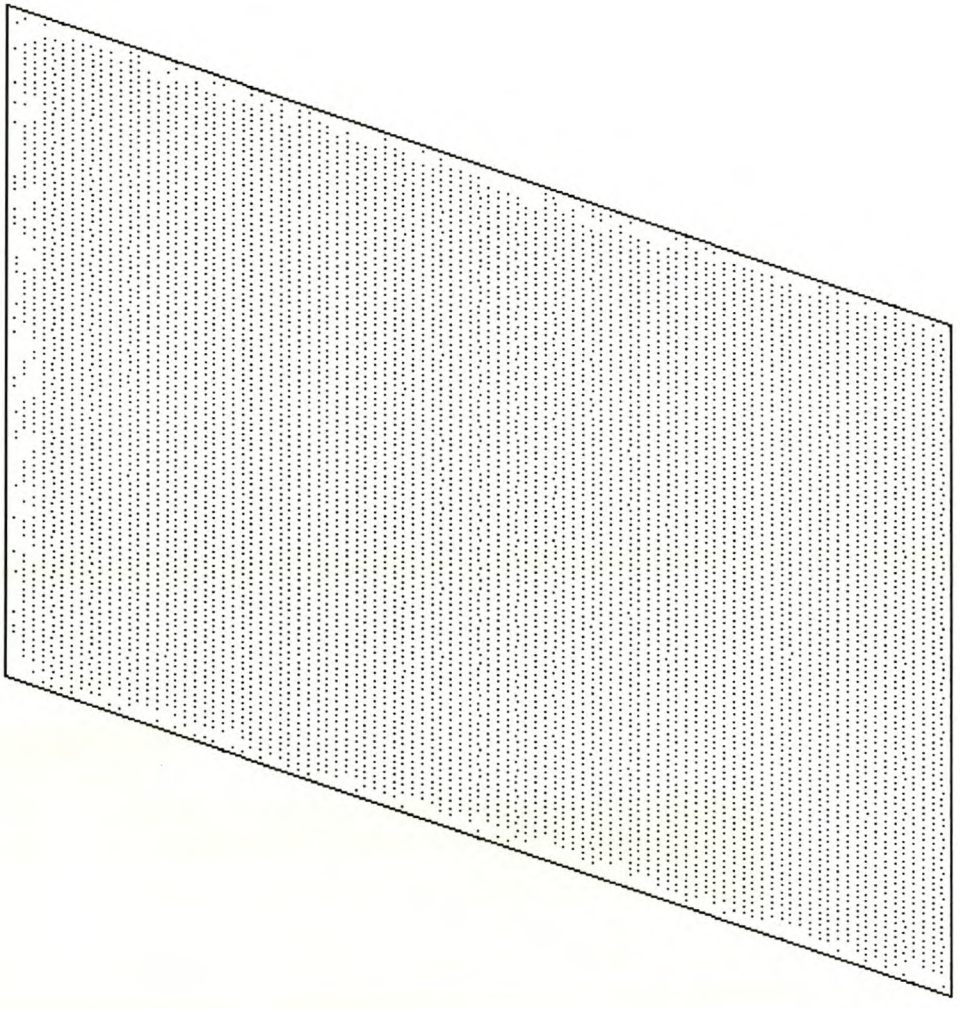

Figure B.17 Extracted plane number 2

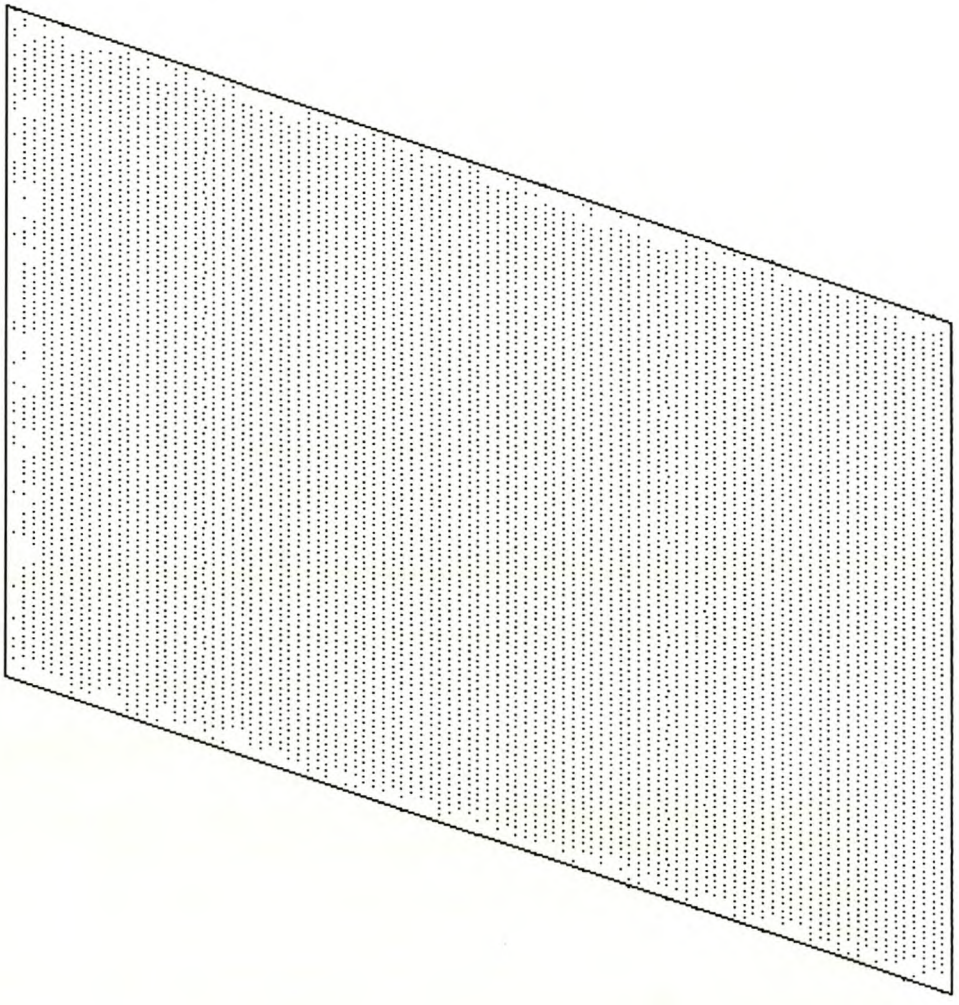

Figure B.18 Extracted plane number 4

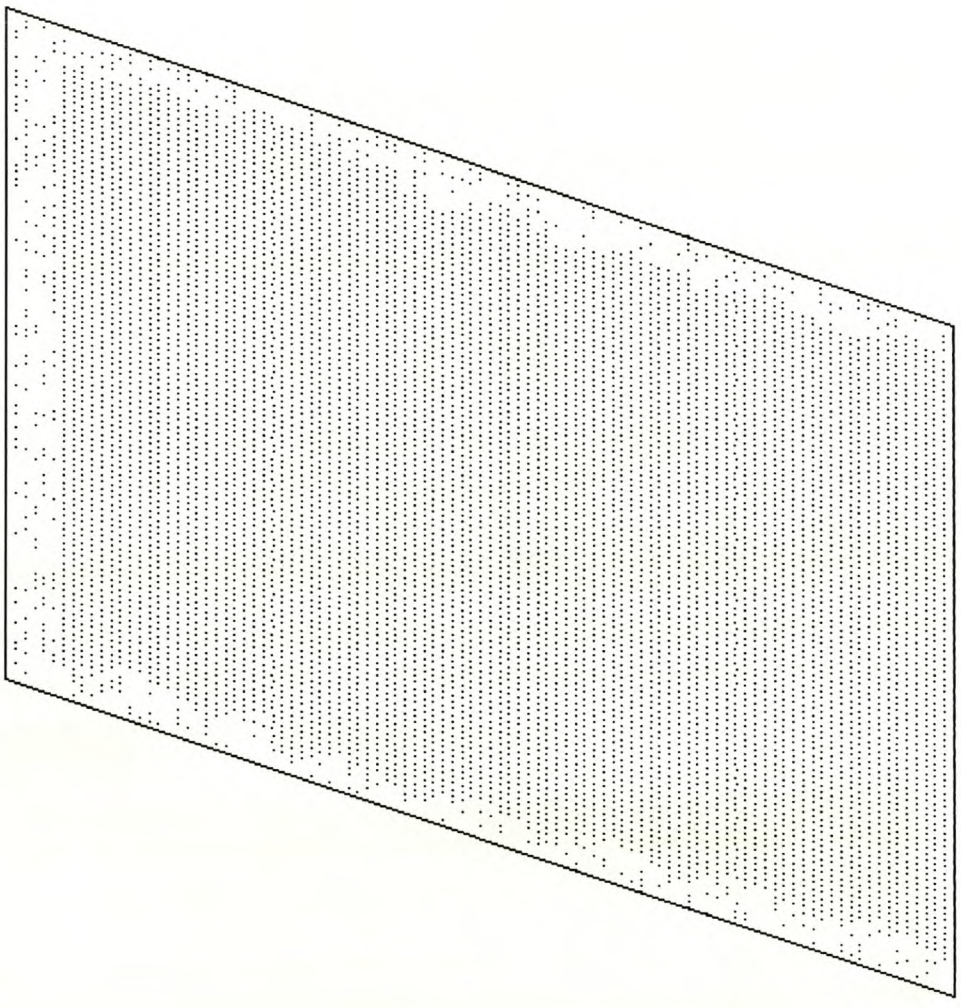

Figure B.19 Extracted plane number 6

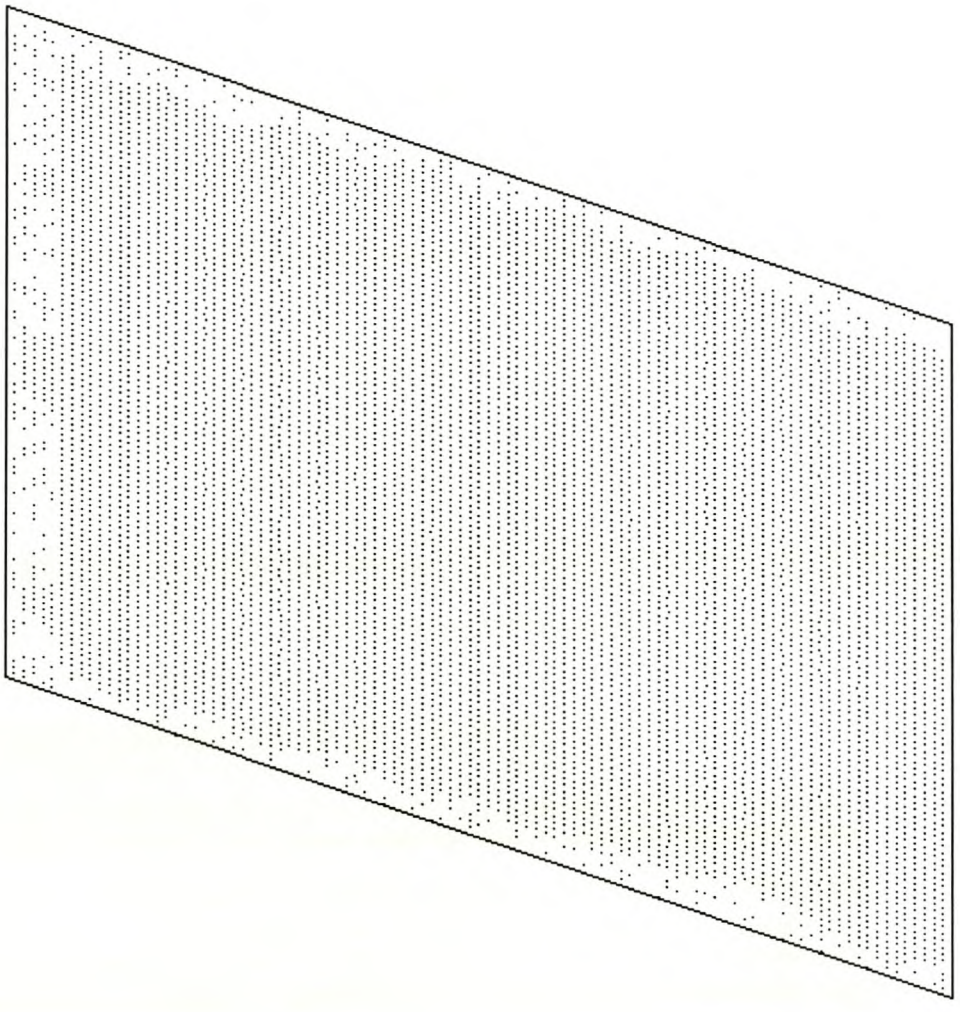

Figure B.20 Extracted plane number 8

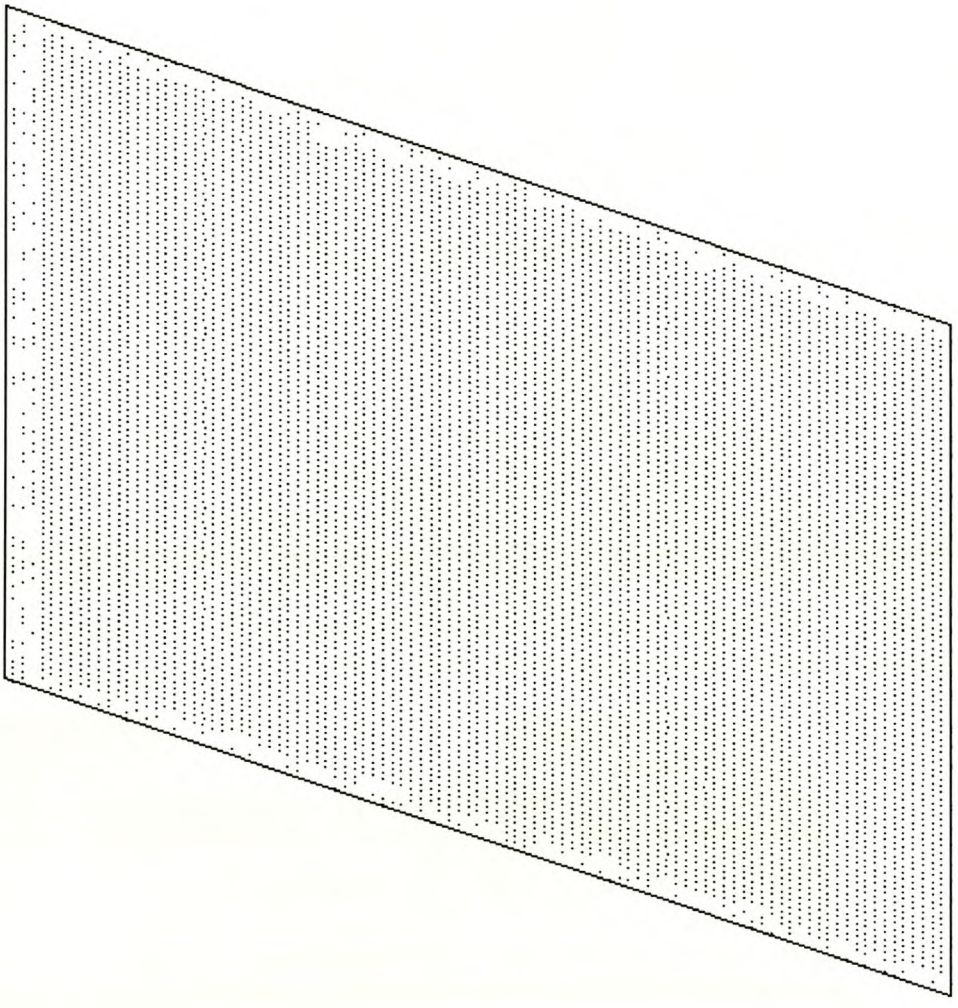

Figure B.21 Extracted plane number 10

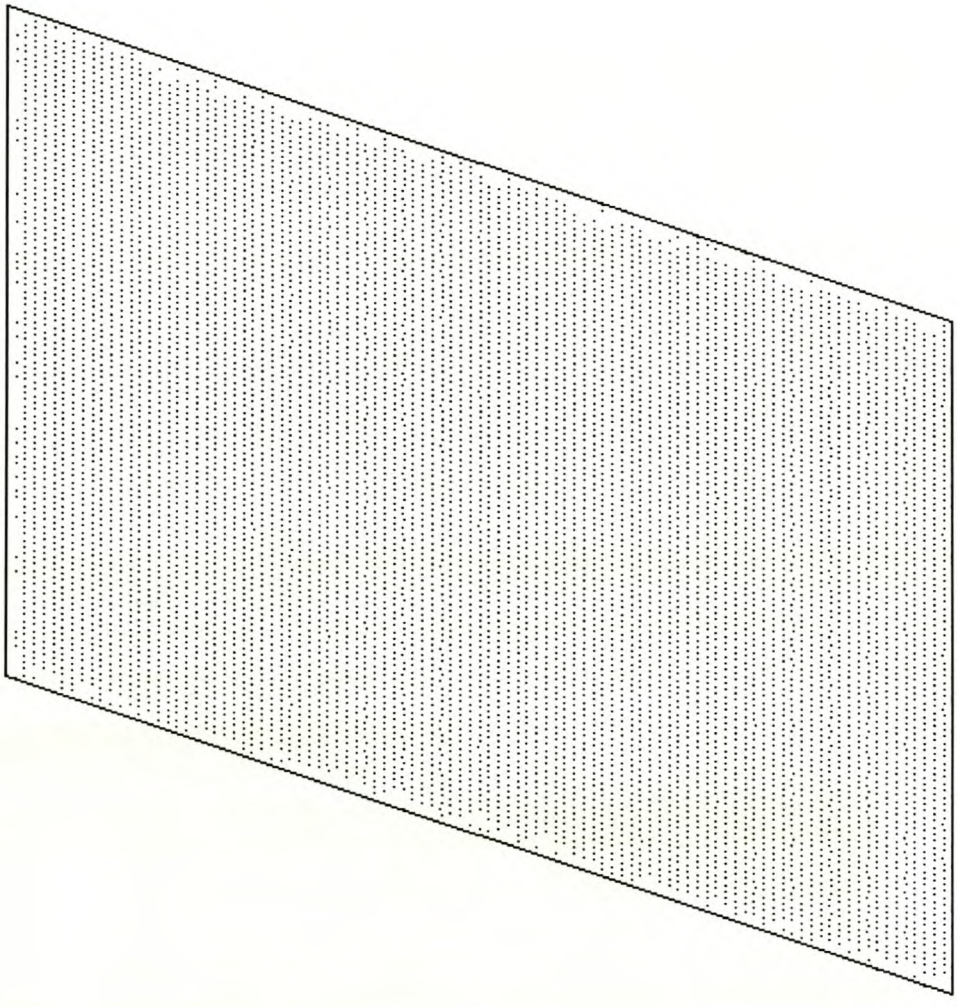

Figure B.22 Extracted plane number 12

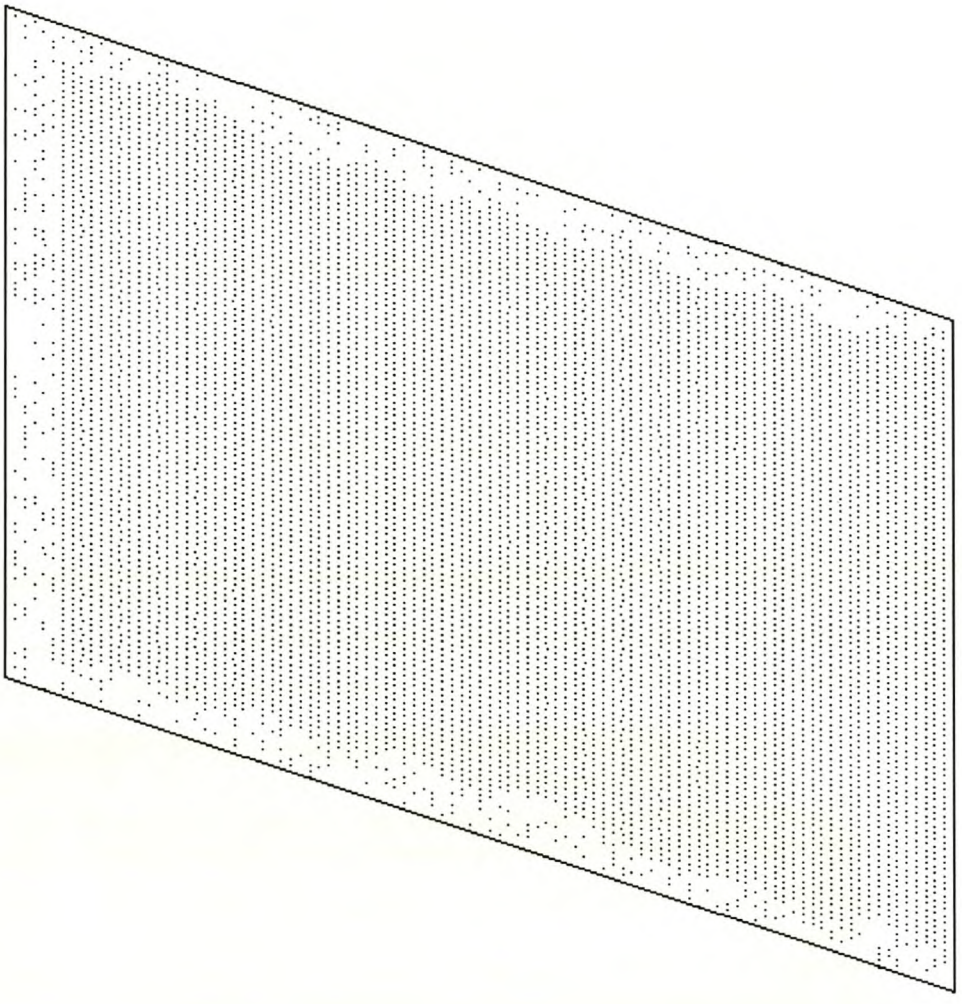

Figure B.23 Extracted plane number 14

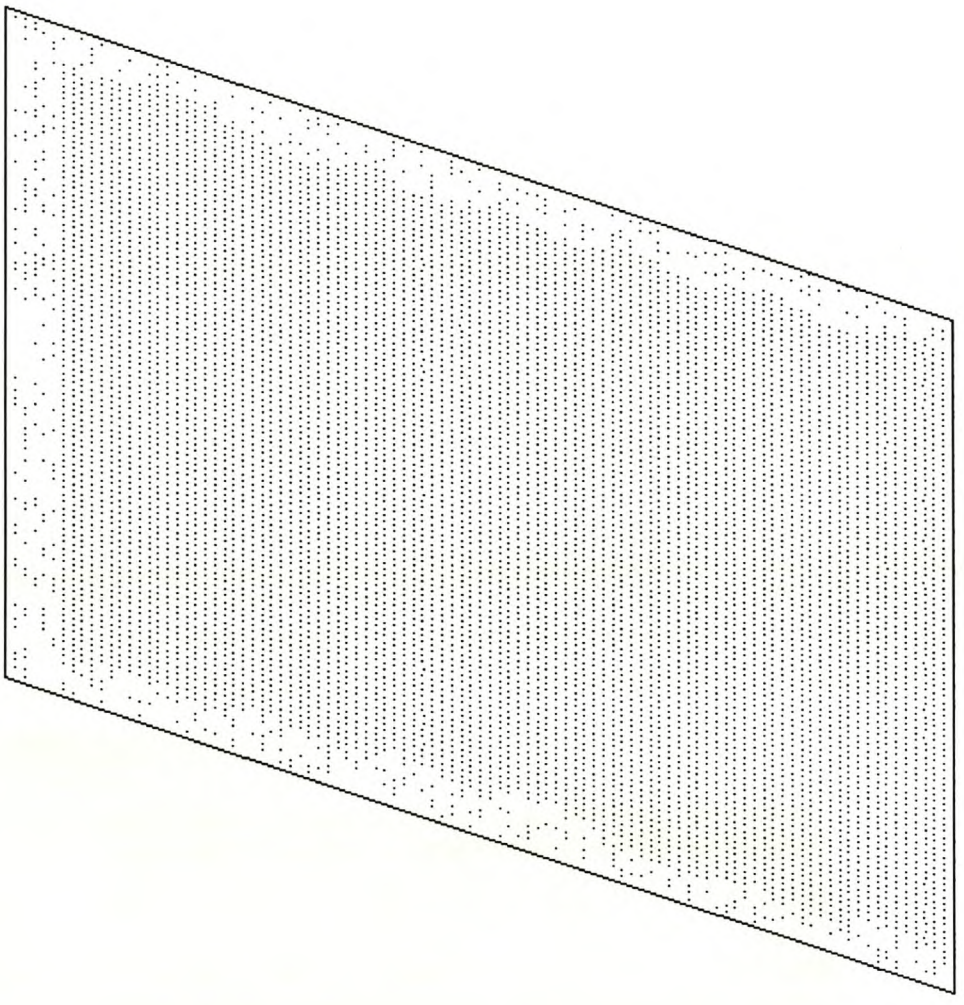

Figure B.24 Extracted plane number 16

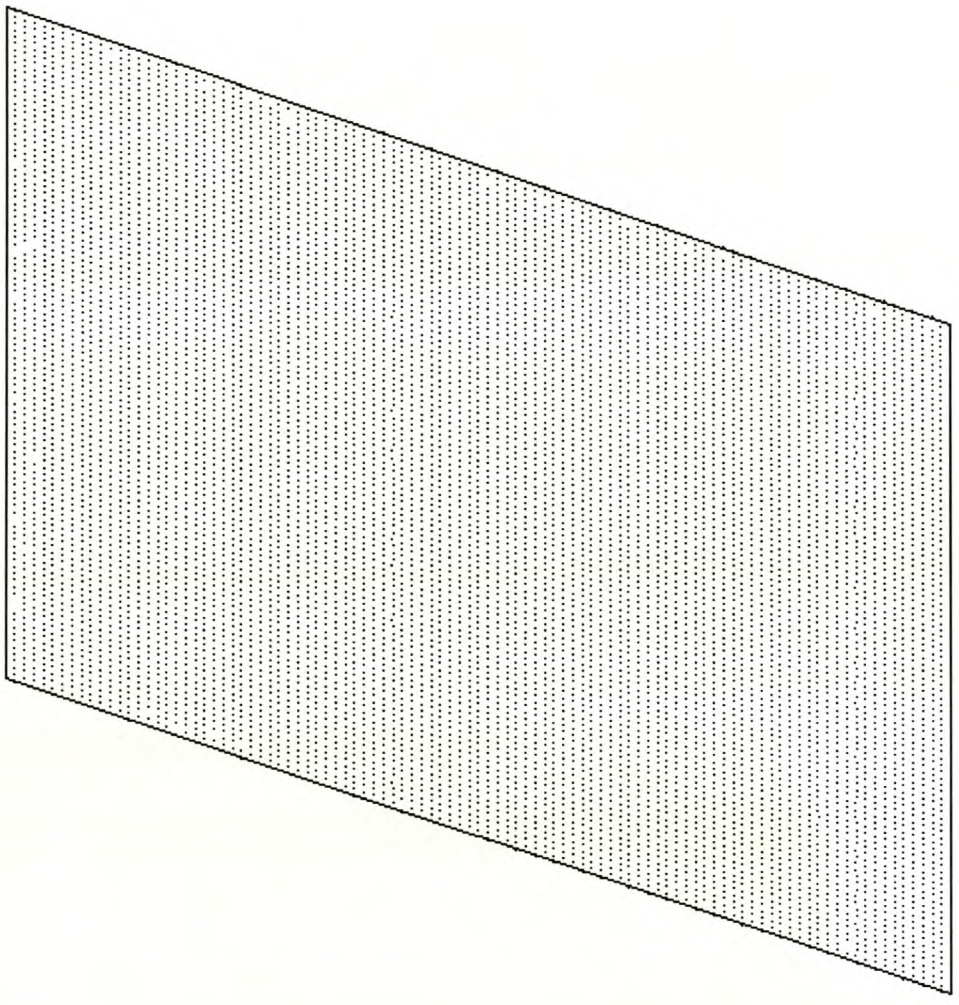

Figure B.25 Extracted plane number 18

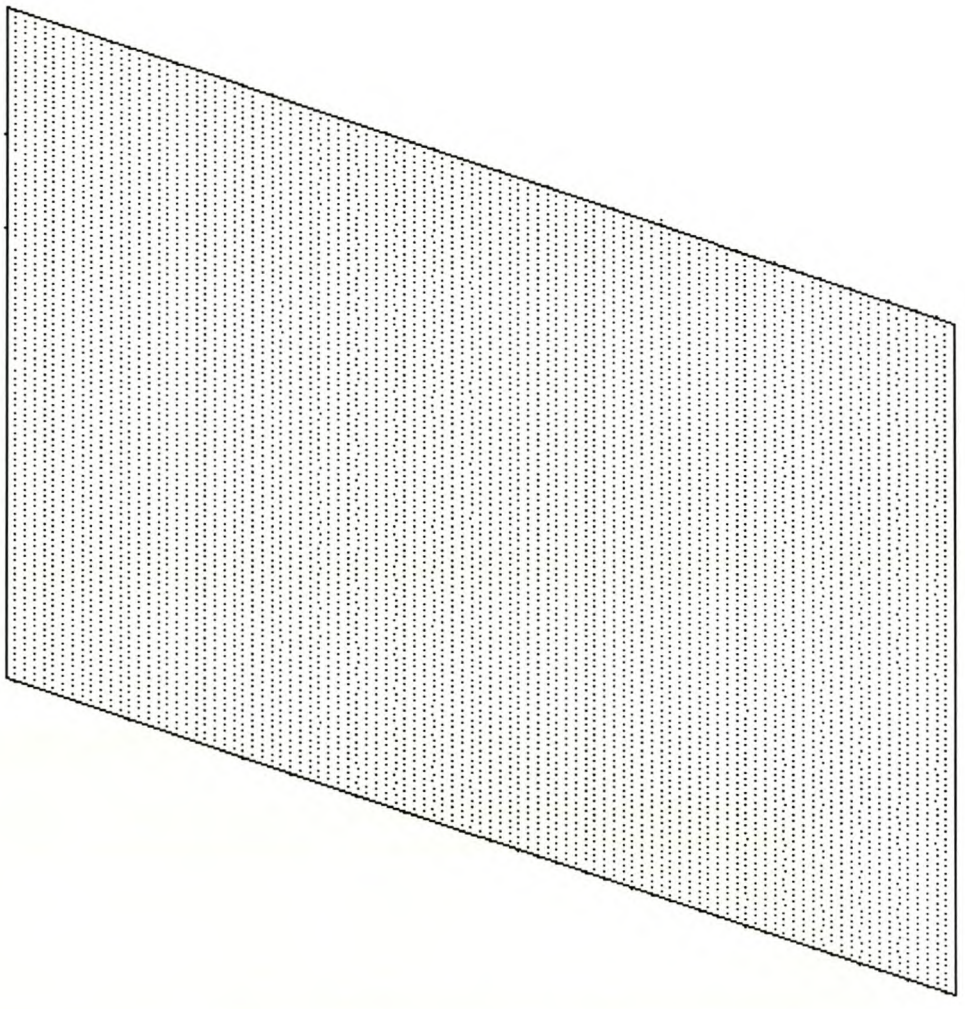

Figure B.26 Extracted plane number 20

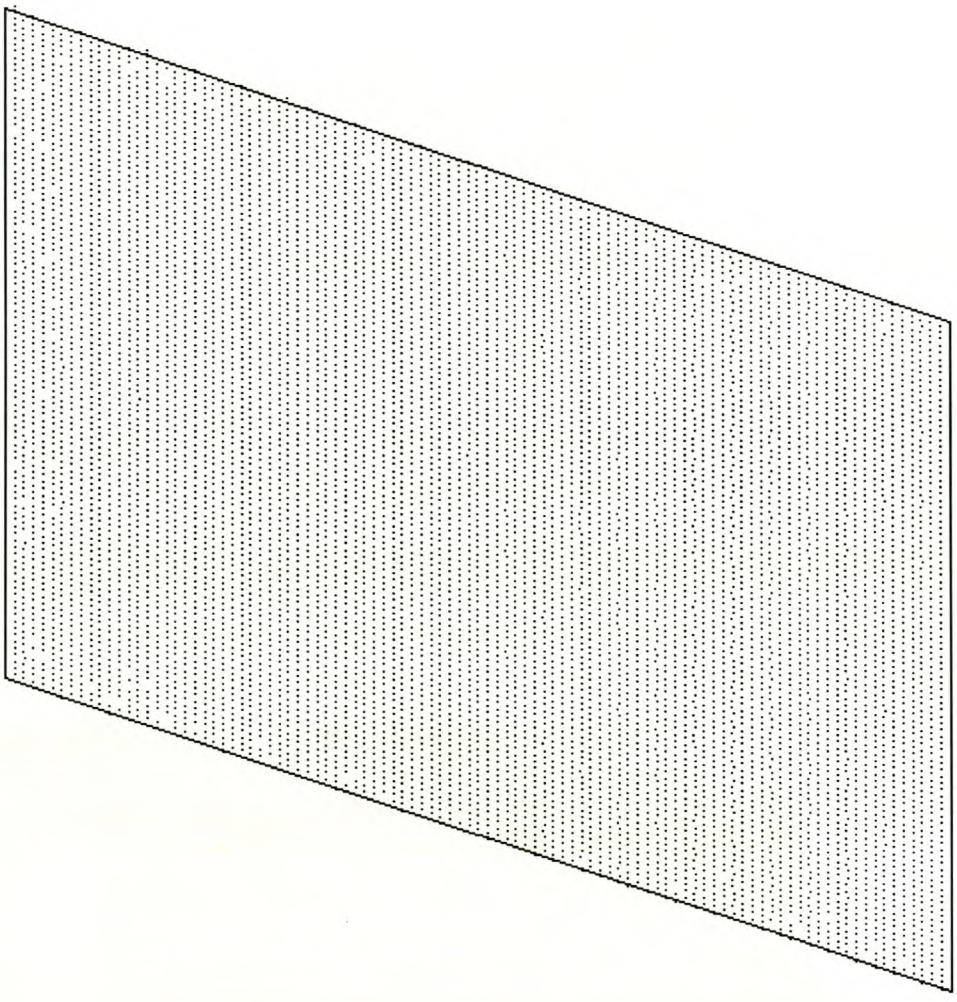

Figure B.27 Extracted plane number 22

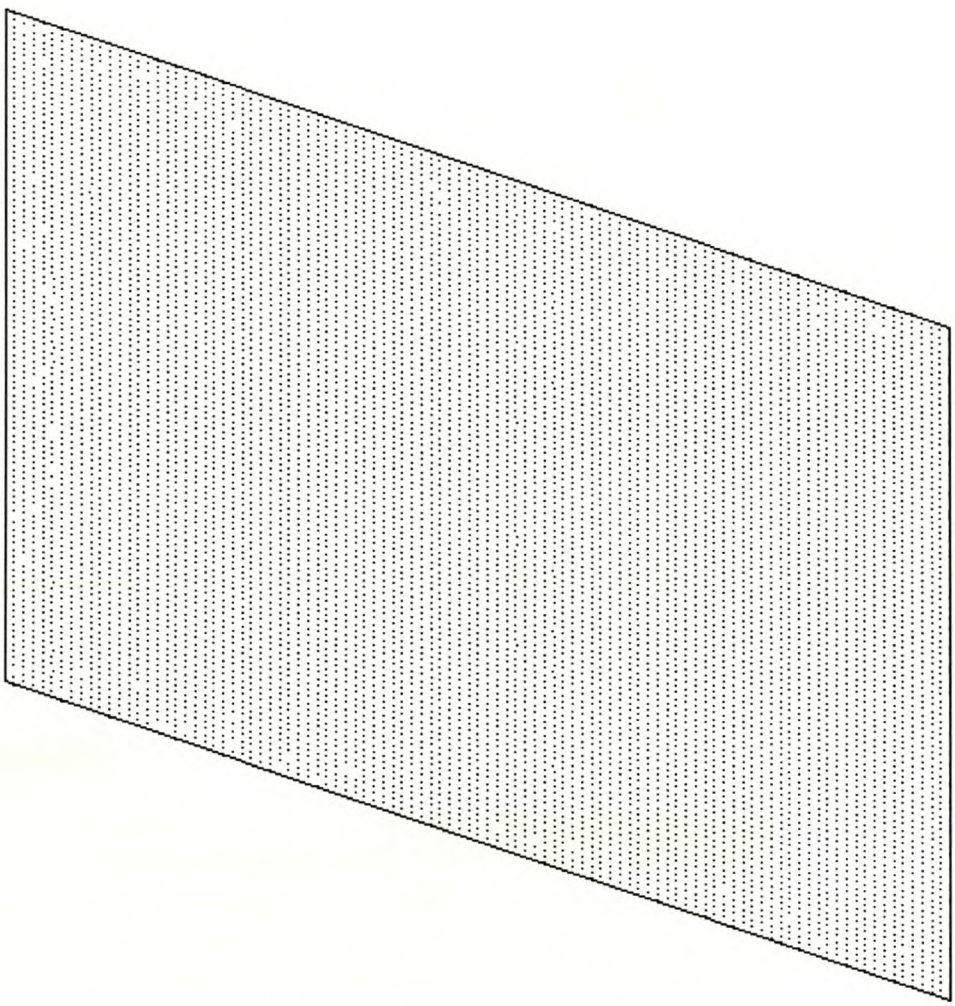

Figure B.28 Extracted plane number 24

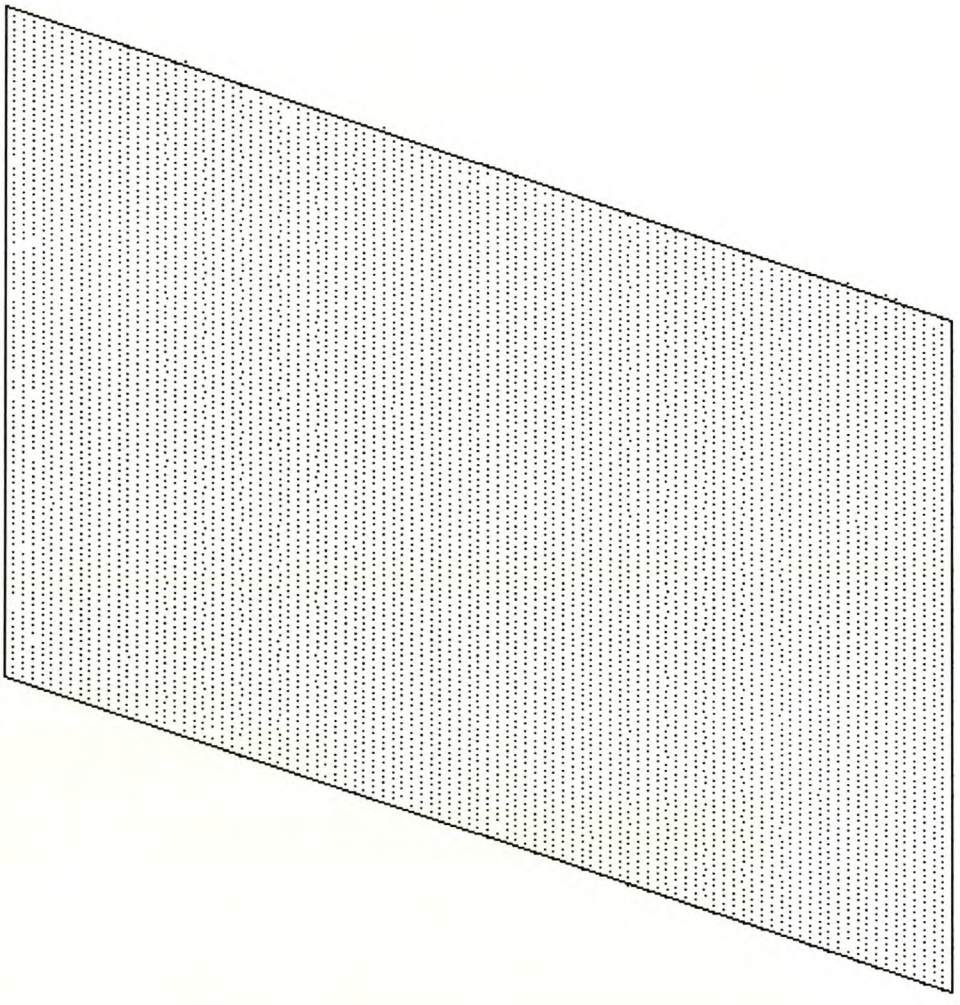

Figure B.29 Extracted plane number 26

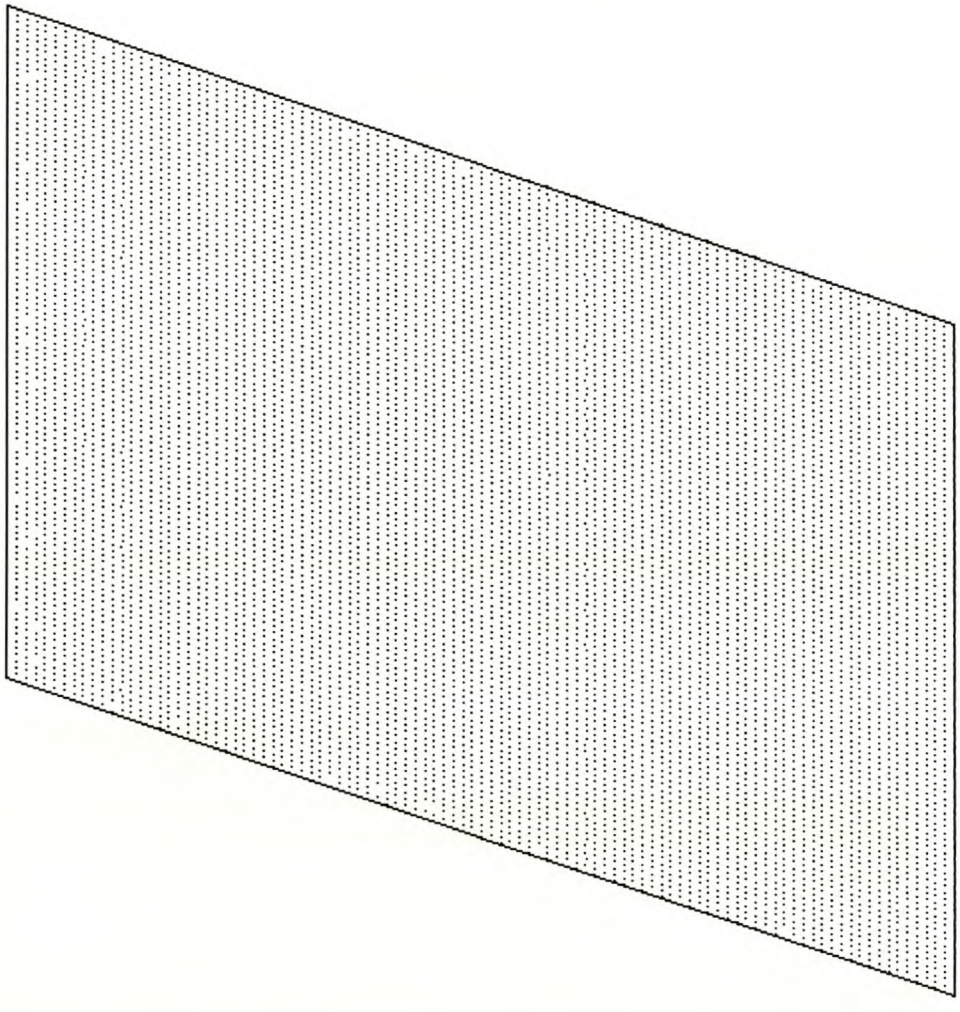

Figure B.30 Extracted plane number 28

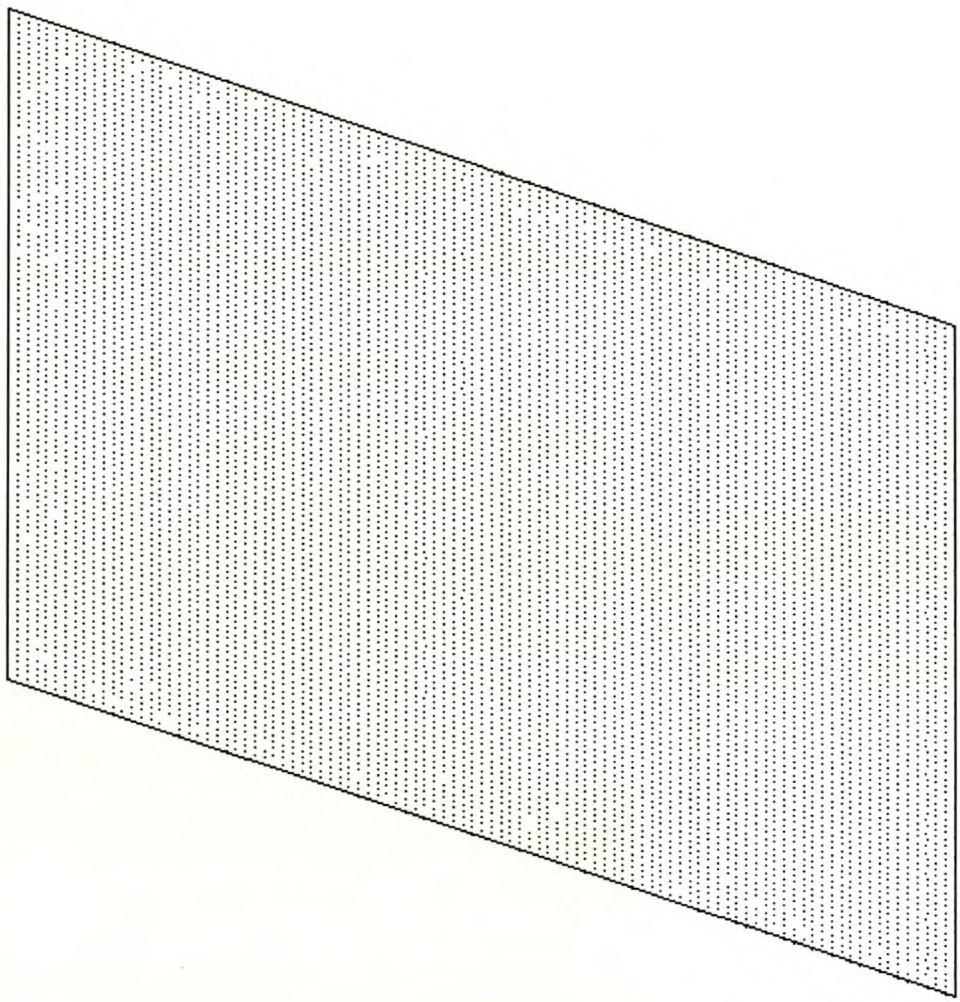

Figure B.31 Extracted plane number 30

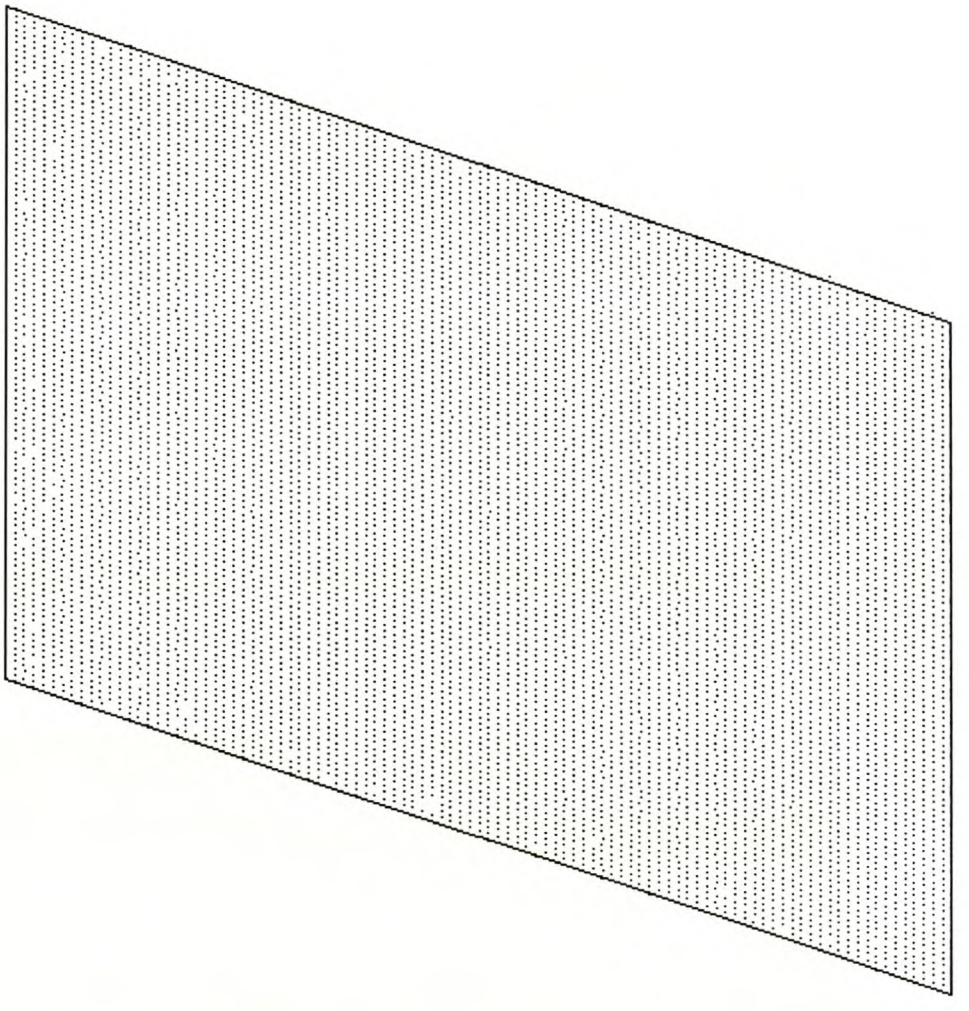

Figure B.32 Extracted plane number 32

B.2 Spheres

This paragraph shows the extracted small spheres (10mm radius) and extracted large spheres (200mm radius) of Chapter 6. Spheres showing the same extraction properties are grouped together.

B.2.1 Small Spheres

The eight spheres (1, 5, 7, 9, 11, 13, 15 and 31) show the same extraction effect where most of the inner points are included but most of the boundary nodes' points are not included. Spheres 1, 9 and 11 have octree nodes that are twice the size of the average point pitch. That is the reason why their boundaries are better found than that of spheres 5, 7, 13, 15 and 31. Points in the boundaries are discarded because the boundary node accuracy is too strict except with sphere 31 for which the node size is three times the average point pitch and the node tolerance equal to 0.7. The reason for the holes in the boundary of sphere 31 is that the node tolerance is strict and the data accuracy is 0.01mm, which means that the boundary nodes are found correctly, but the points in the boundary node are discarded because the boundary node accuracy is half of the data accuracy (0.005mm).

The three spheres (3, 23 and 27) show the same extraction effect where most of the inner points are included but points outside the sphere boundaries are also included. Spheres 3, 23 and 27 show this result as their fit accuracy (0.15mm) is set to three times the data accuracy (0.05). Some points are included outside the boundary because these set values has the effect to overgrow the boundary slightly.

Spheres 21, 25 and 29 can be considered well fitted spheres as only few points are discarded that belongs to the boundary and one or two points outside the boundary are included. Spheres 17 and 19 can be considered as the best fitted spheres.

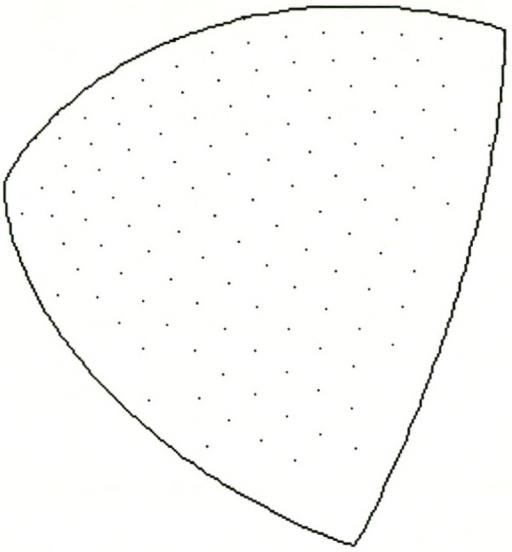

Figure B.33 Extracted sphere number 1

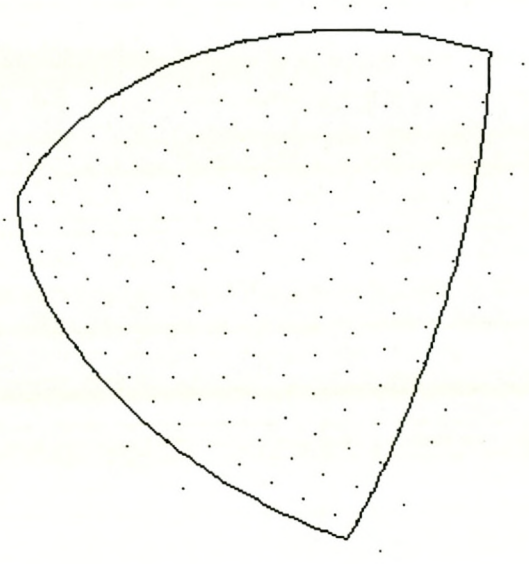

Figure B.34 Extracted sphere number 3

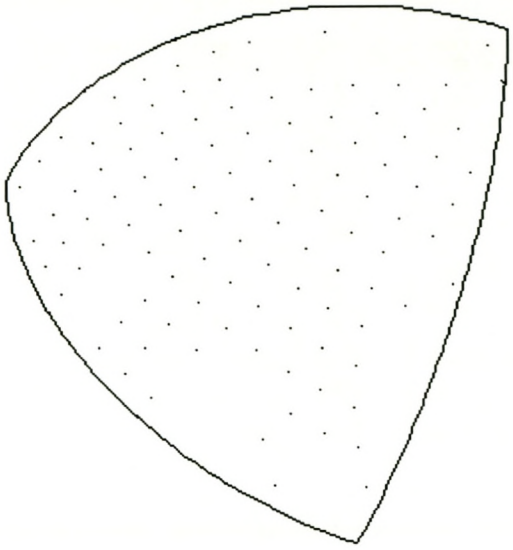

Figure B.35 Extracted sphere number 5

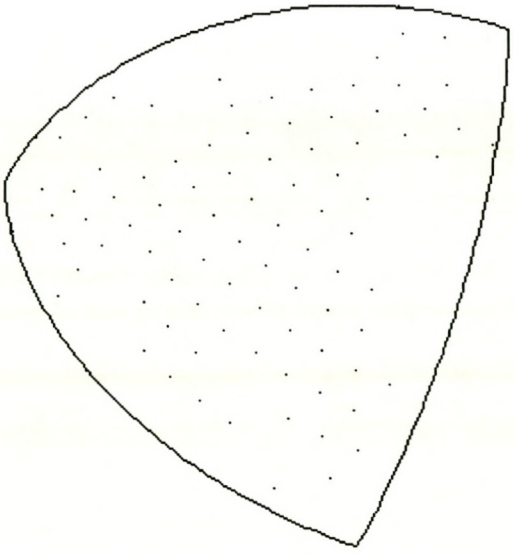

Figure B.36 Extracted sphere number 7

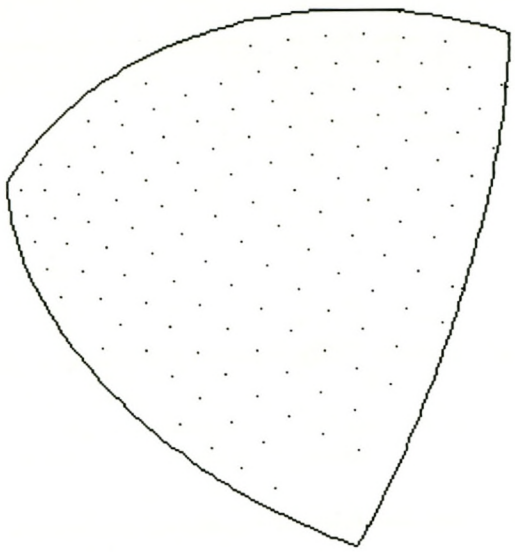

Figure B.37 Extracted sphere number 9

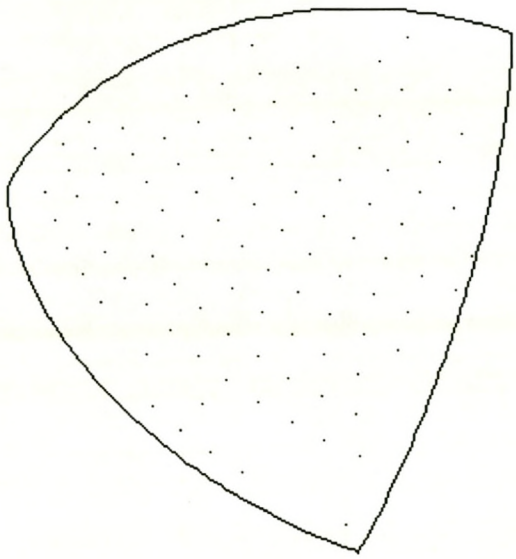

Figure B.38 Extracted sphere number 11

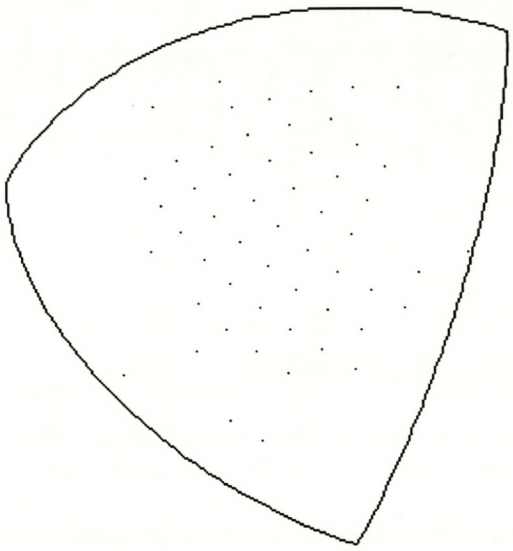

Figure B.39 Extracted sphere number 13

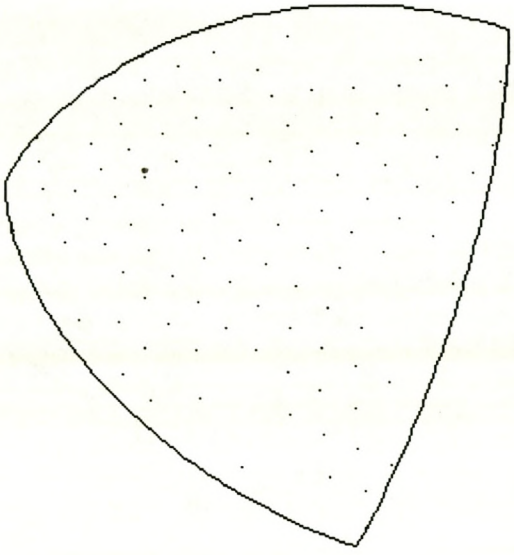

Figure B.40 Extracted sphere number 15

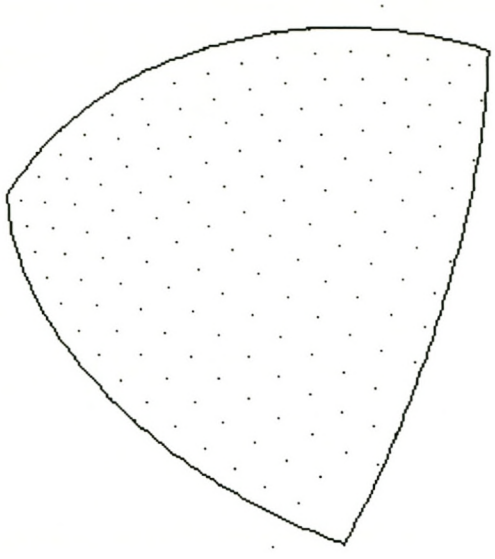

Figure B.41 Extracted sphere number 17

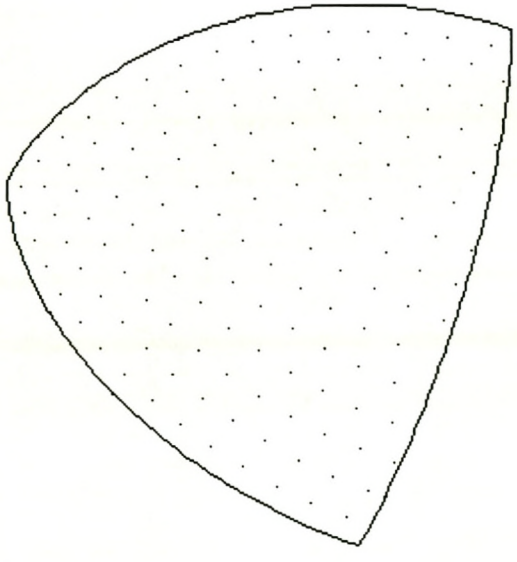

Figure B.42 Extracted sphere number 19

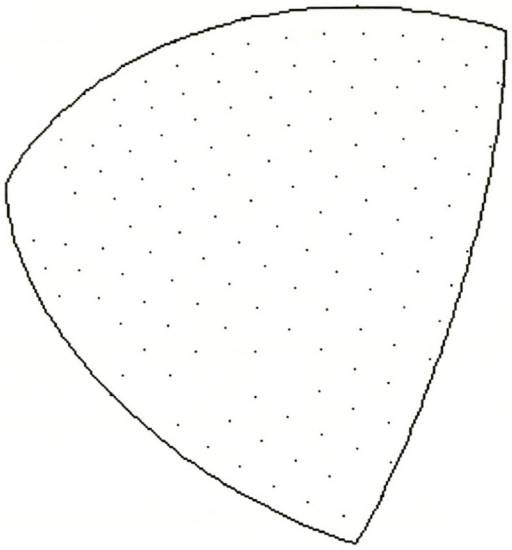

Figure B.43 Extracted sphere number 21

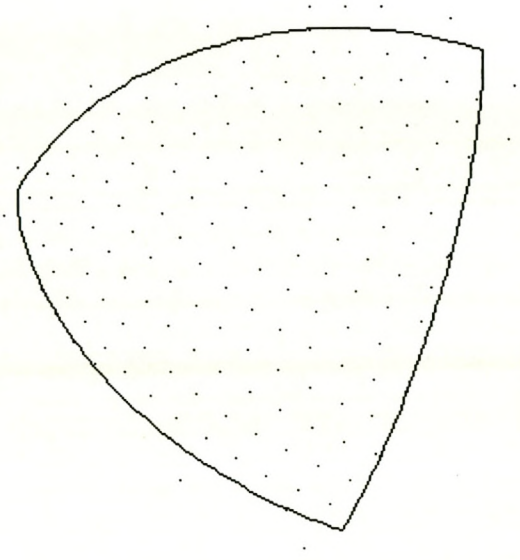

Figure B.44 Extracted sphere number 23

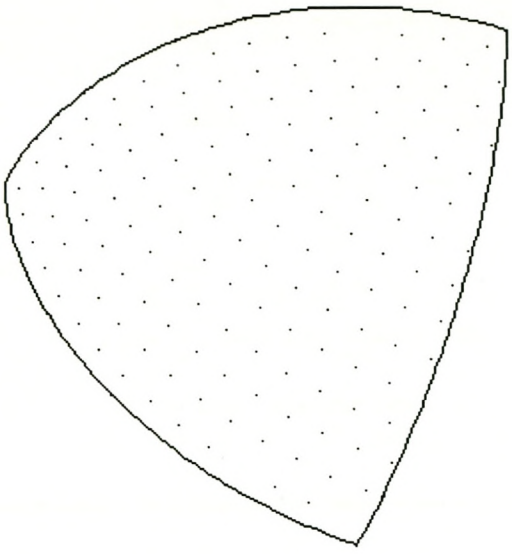

Figure B.45 Extracted sphere number 25

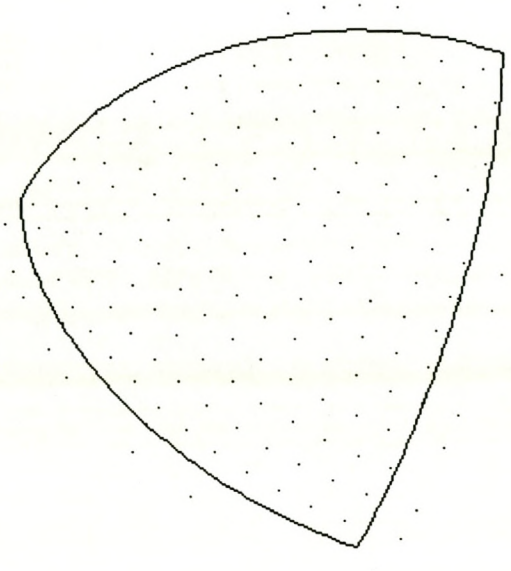

Figure B.46 Extracted sphere number 27

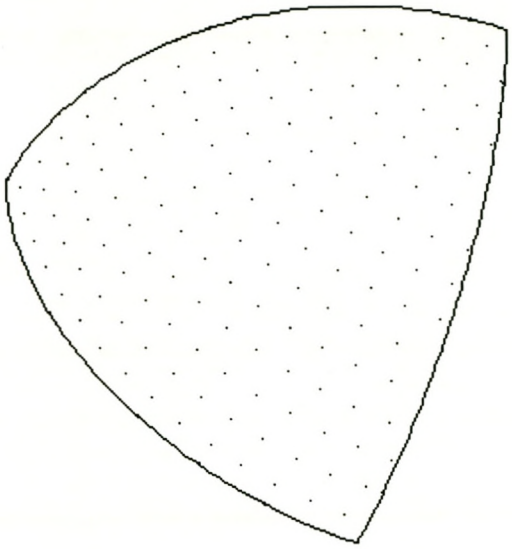

Figure B.47 Extracted sphere number 29

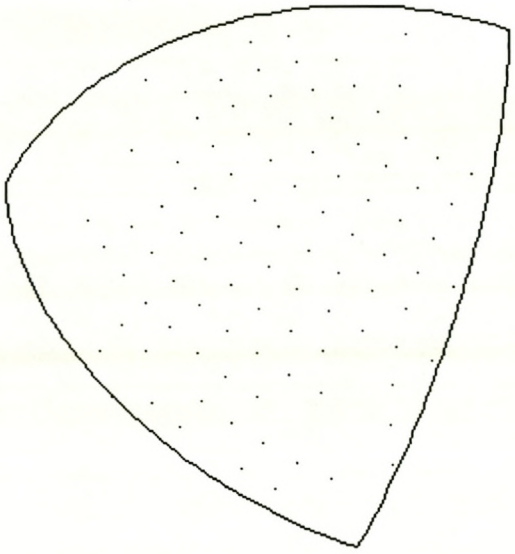

Figure B.48 Extracted sphere number 31

B.2.2 Large Spheres

The three spheres (6, 10 and 16) show the same extraction effect where most of the inner points are included but most of the boundary nodes' points are not included. Spheres 6 and 10 have tight fit accuracies (equal to the data accuracy) as well as tight boundary node accuracies (equal to a tenth of the data accuracy). Sphere 10 has a smaller node size than sphere 6 and therefore the result looks better. Sphere 16 has a tight node tolerance (0.7) as well as a tight boundary node accuracy (one tenth of the data accuracy).

The seven spheres (2, 8, 12, 20, 22, 26 and 32) show the same extraction effect where most of the inner points are included but points outside the sphere boundaries are also included. The reasons vary from a small node size (twice the average point pitch) with a node tolerance of 0.5 to spheres with a large fit accuracy (three times the data accuracy) or spheres with a large boundary node accuracy (half the data accuracy).

The six spheres (4, 14, 18, 24, 28 and 30) can be considered as well fitted spheres with the best ones being spheres 4, 28 and 30. The reason is that some points inside the boundary may be discarded and some points outside the boundary included giving an average boundary. This is shown for spheres 14, 18 and 24. Sphere 14 shows this phenomenon the best and the reasons are a tight fit accuracy (same as the data accuracy), a tight node tolerance (0.7), a tight boundary node accuracy, but a larger node size (three times the average point pitch) and a data accuracy of 0.05mm.

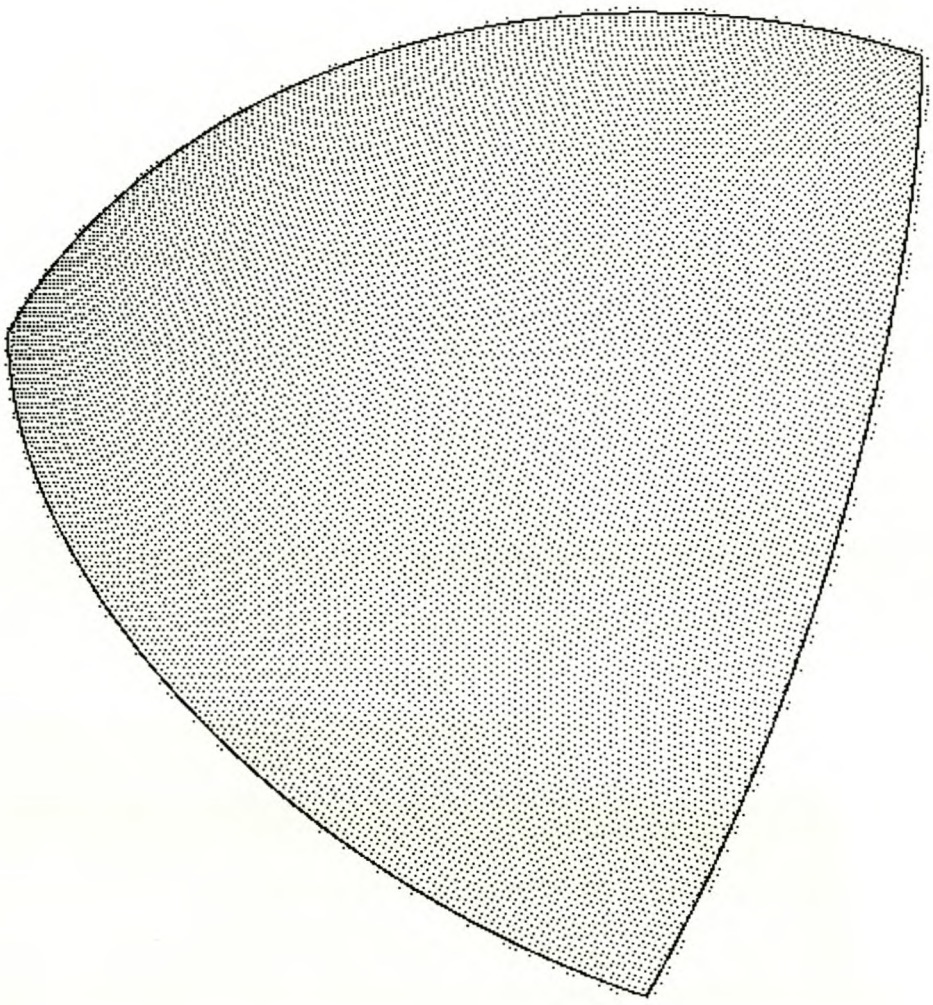

Figure B.49 Extracted sphere number 2

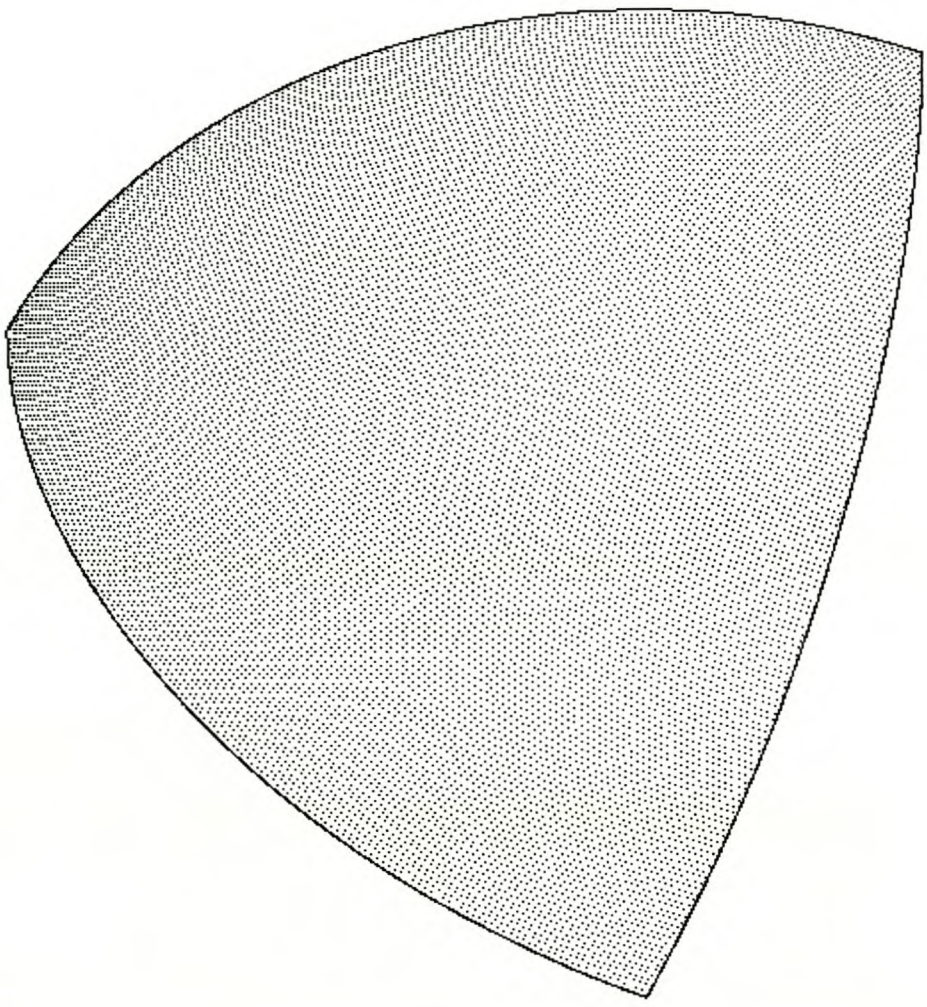

Figure B.50 Extracted sphere number 4

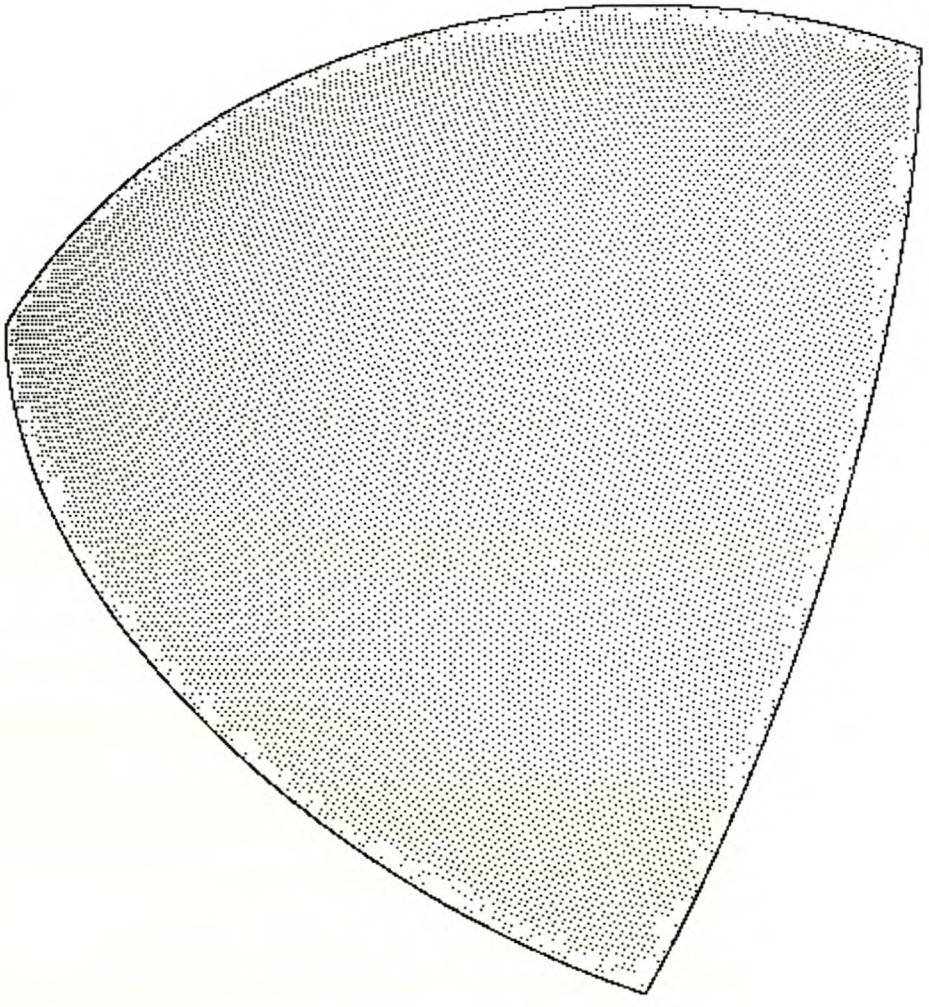

Figure B.51 Extracted sphere number 6

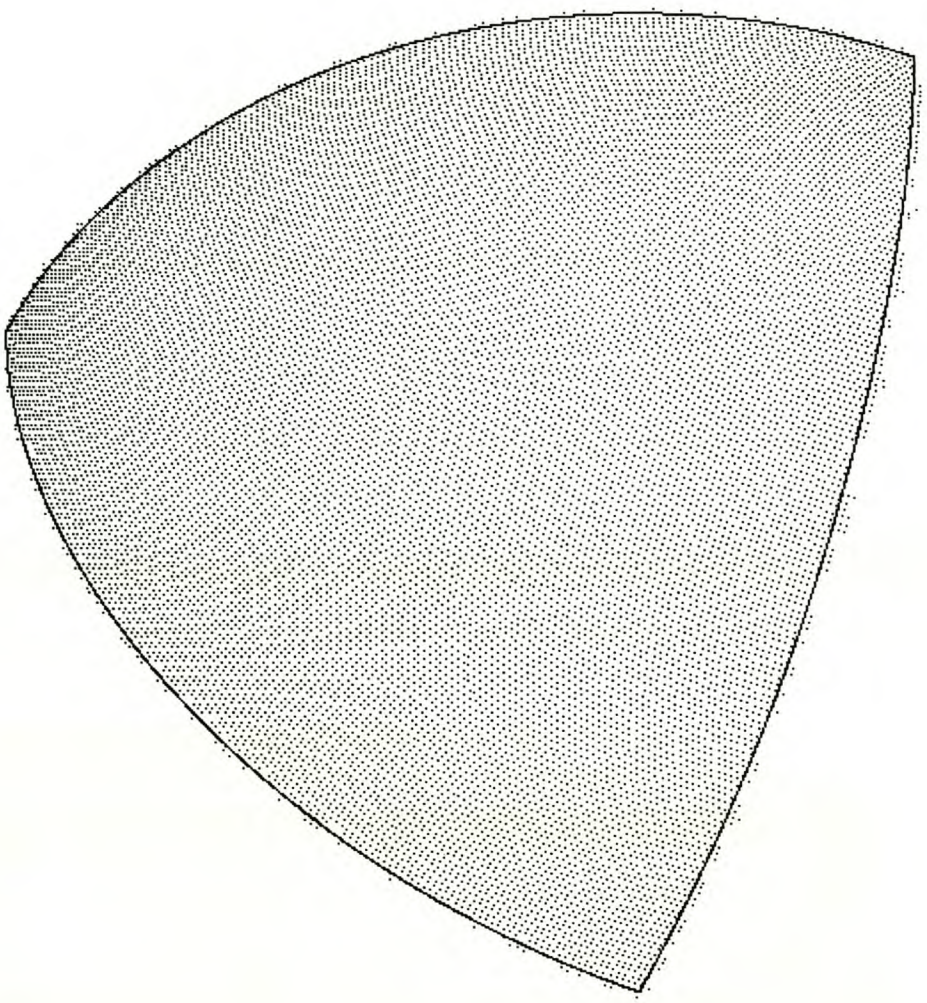

Figure B.52 Extracted sphere number 8

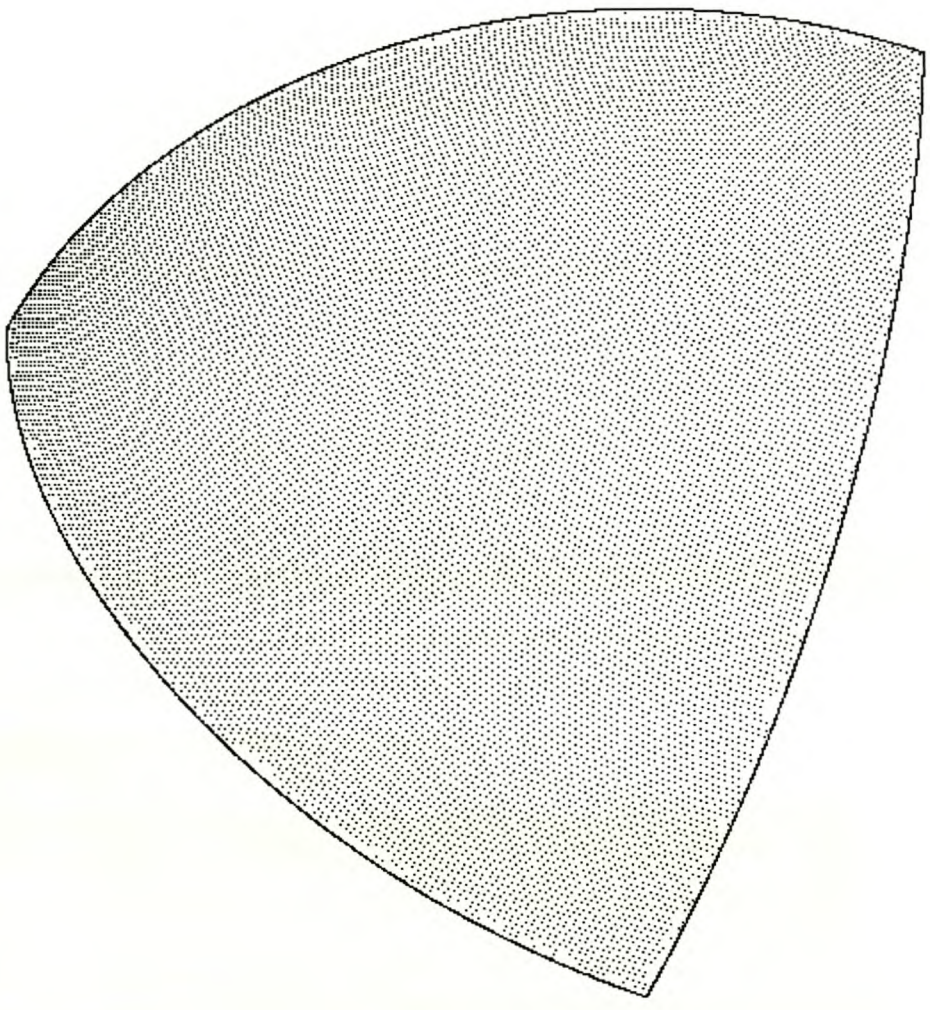

Figure B.53 Extracted sphere number 10

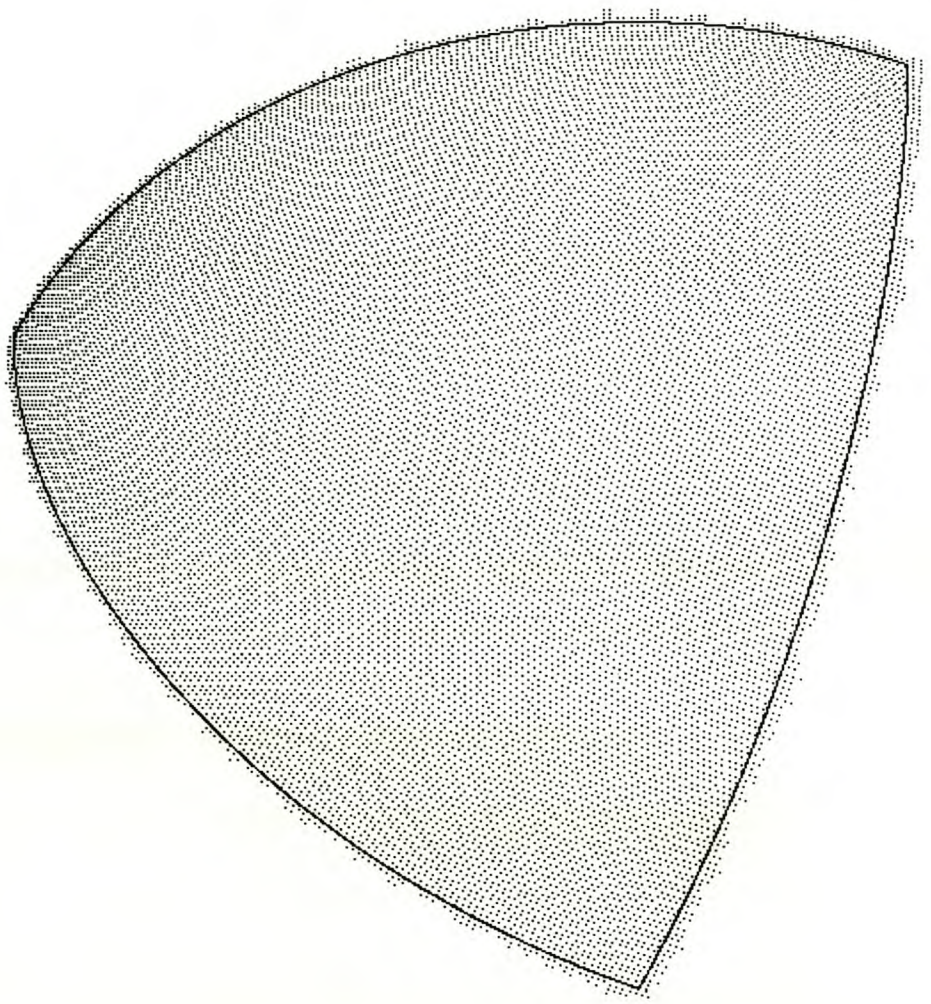

Figure B.54 Extracted sphere number 12

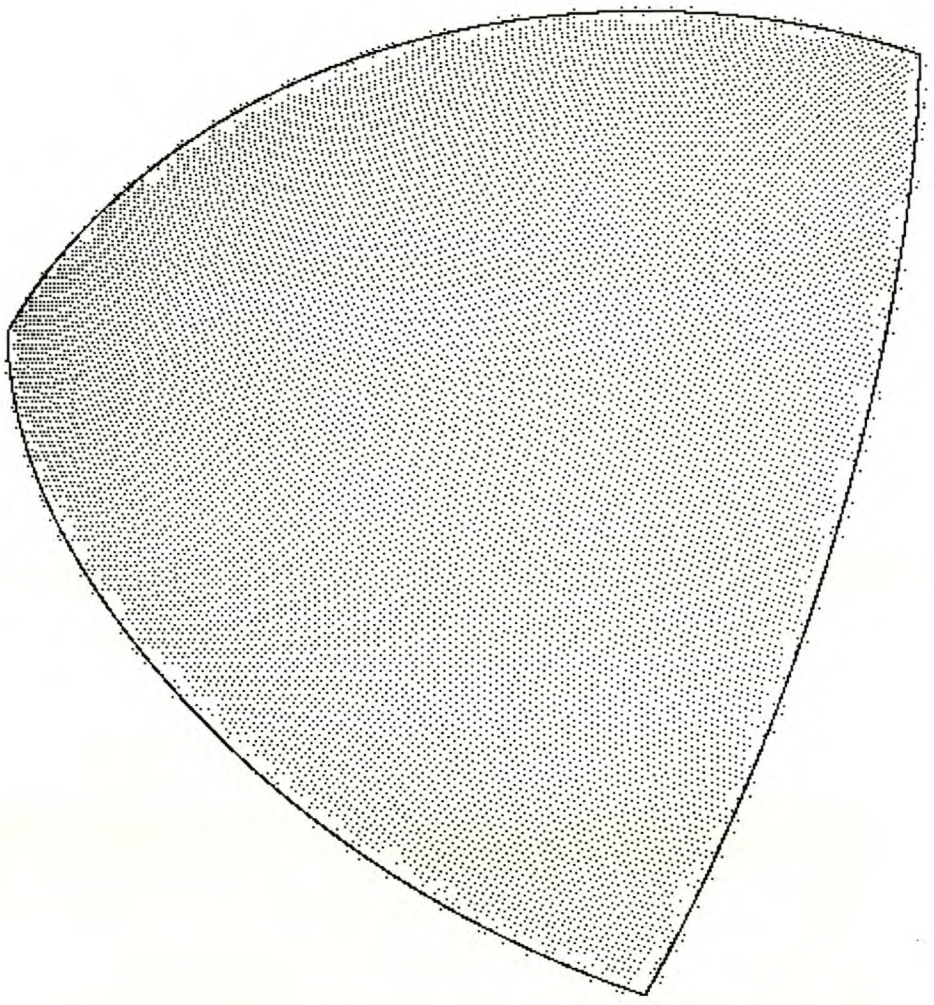

Figure B.55 Extracted sphere number 14

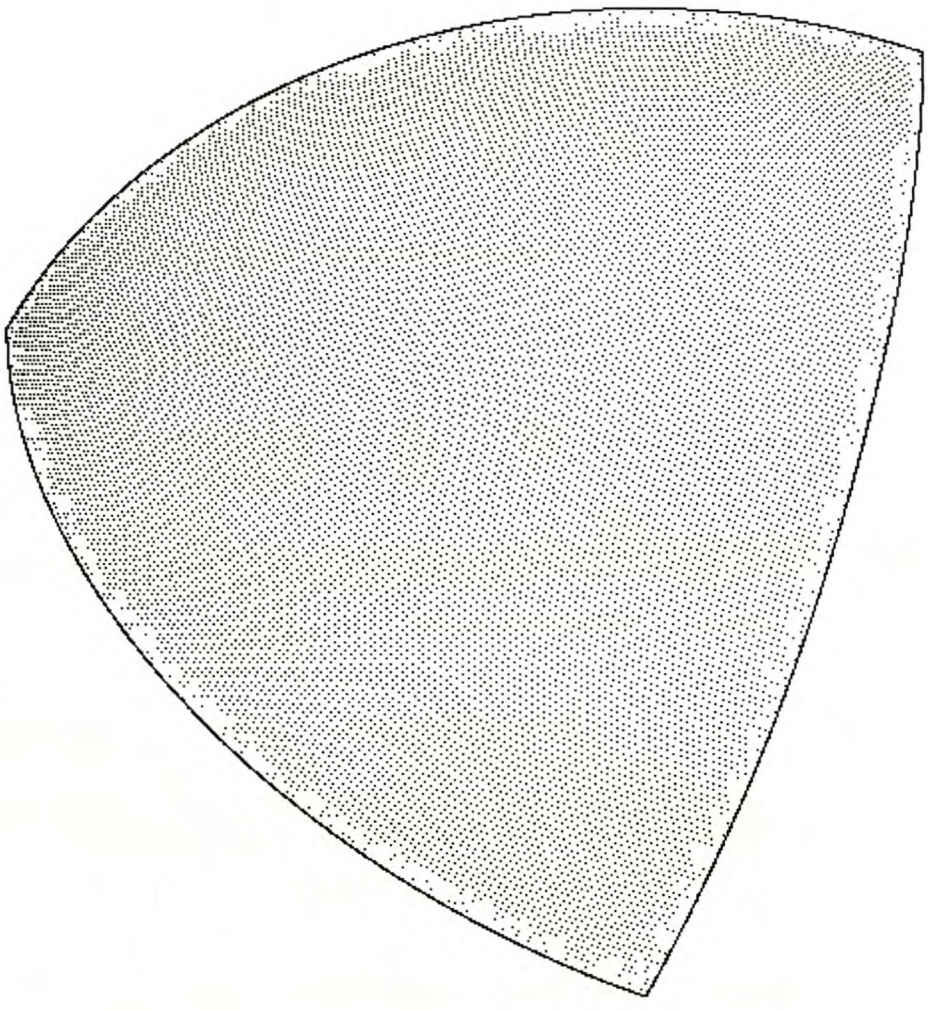

Figure B.56 Extracted sphere number 16

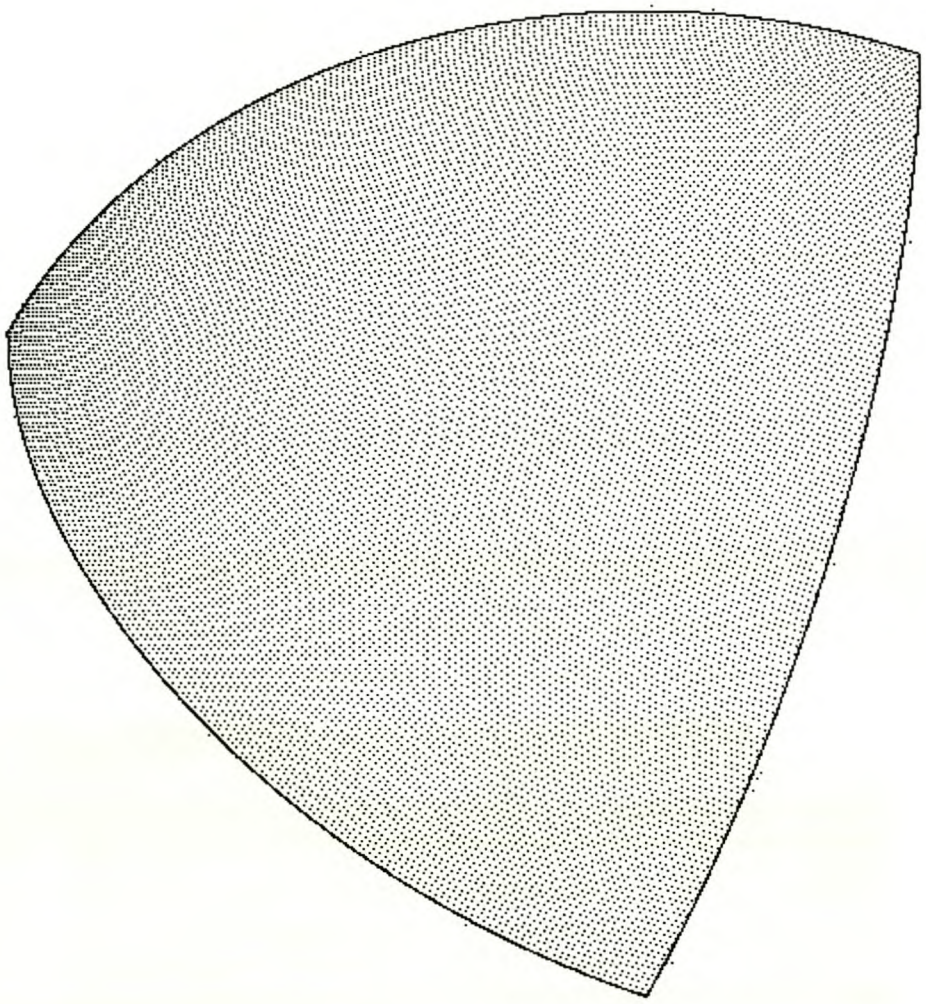

Figure B.57 Extracted sphere number 18

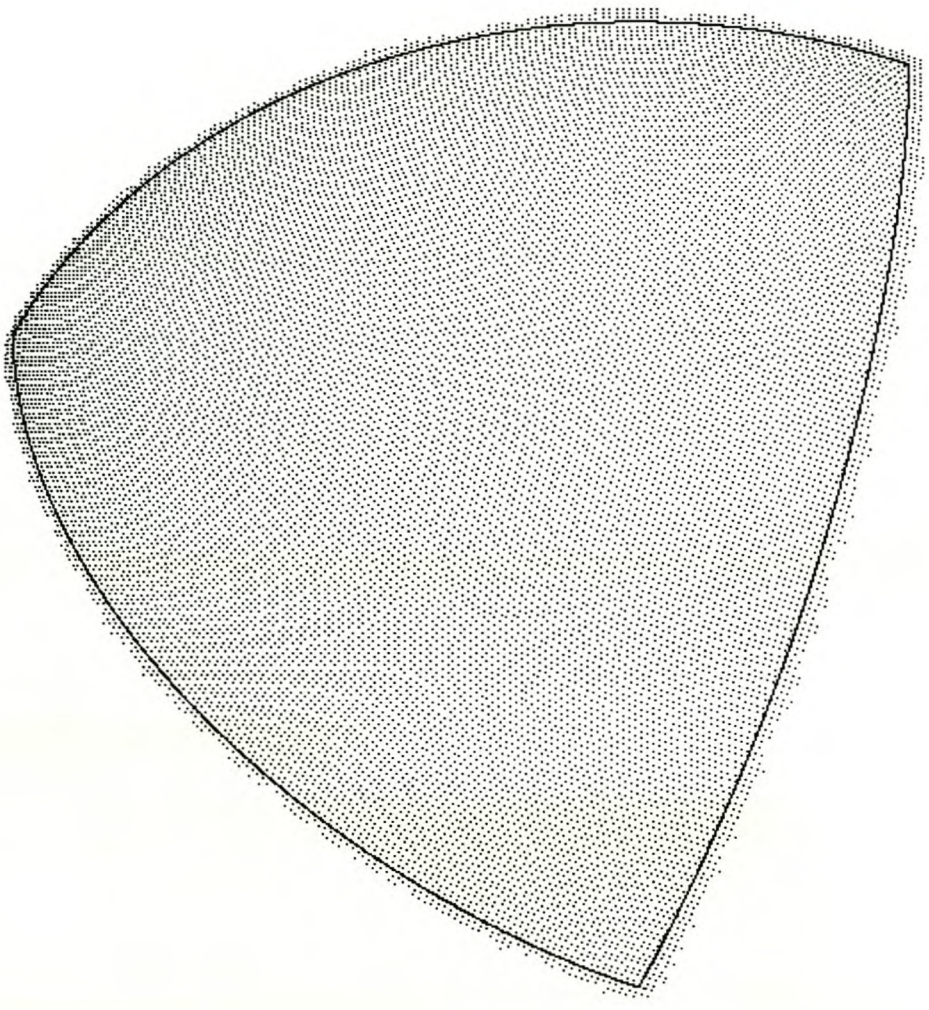

Figure B.58 Extracted sphere number 20

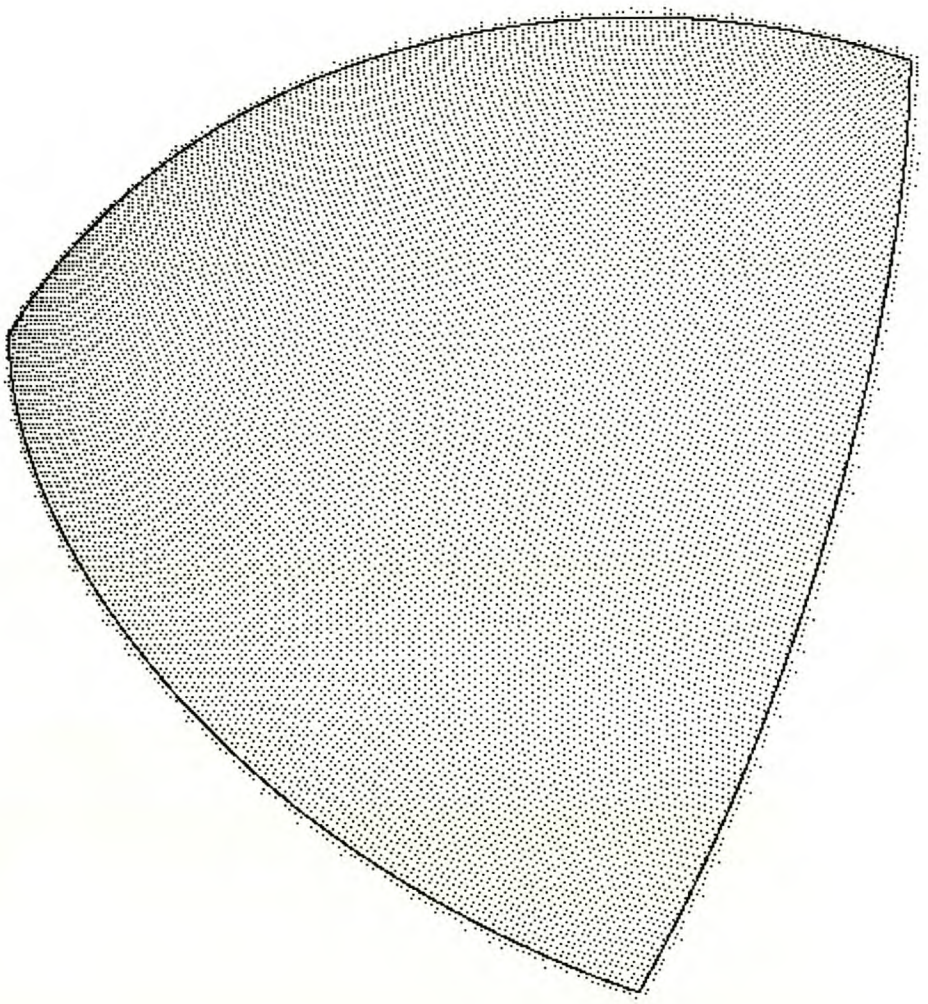

Figure B.59 Extracted sphere number 22

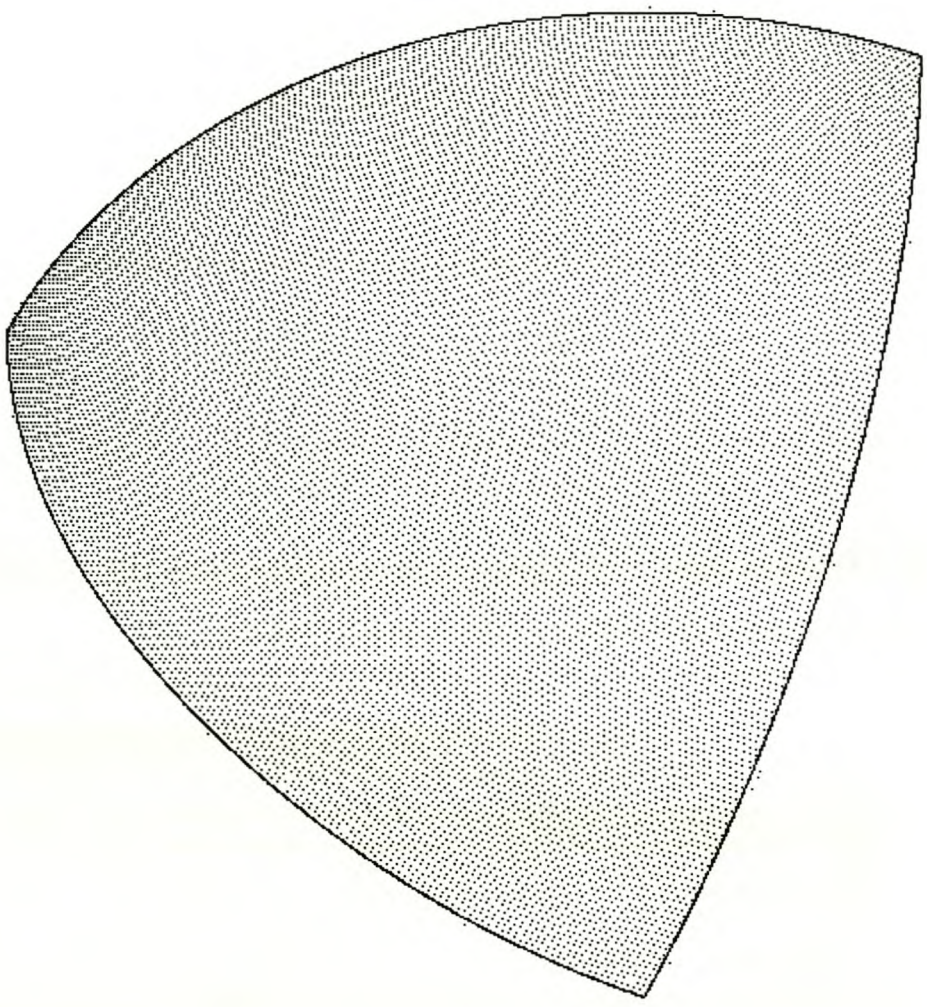

Figure B.60 Extracted sphere number 24

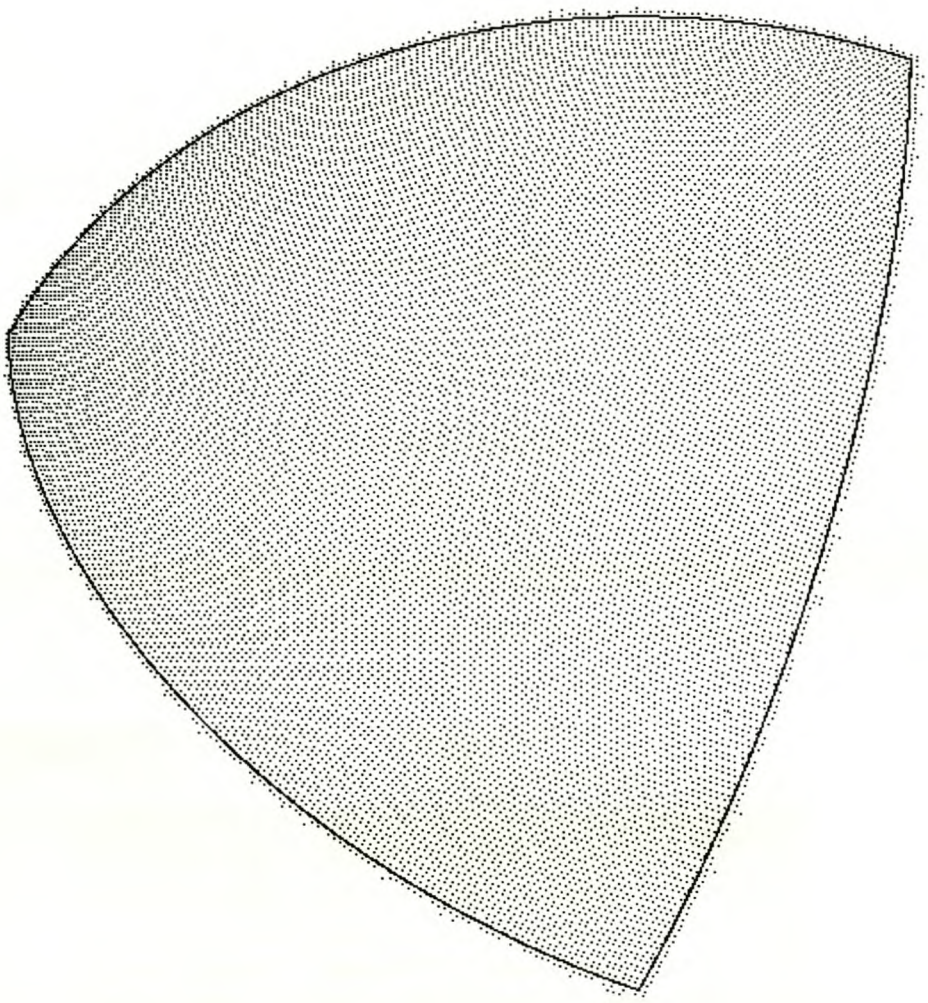

Figure B.61 Extracted sphere number 26

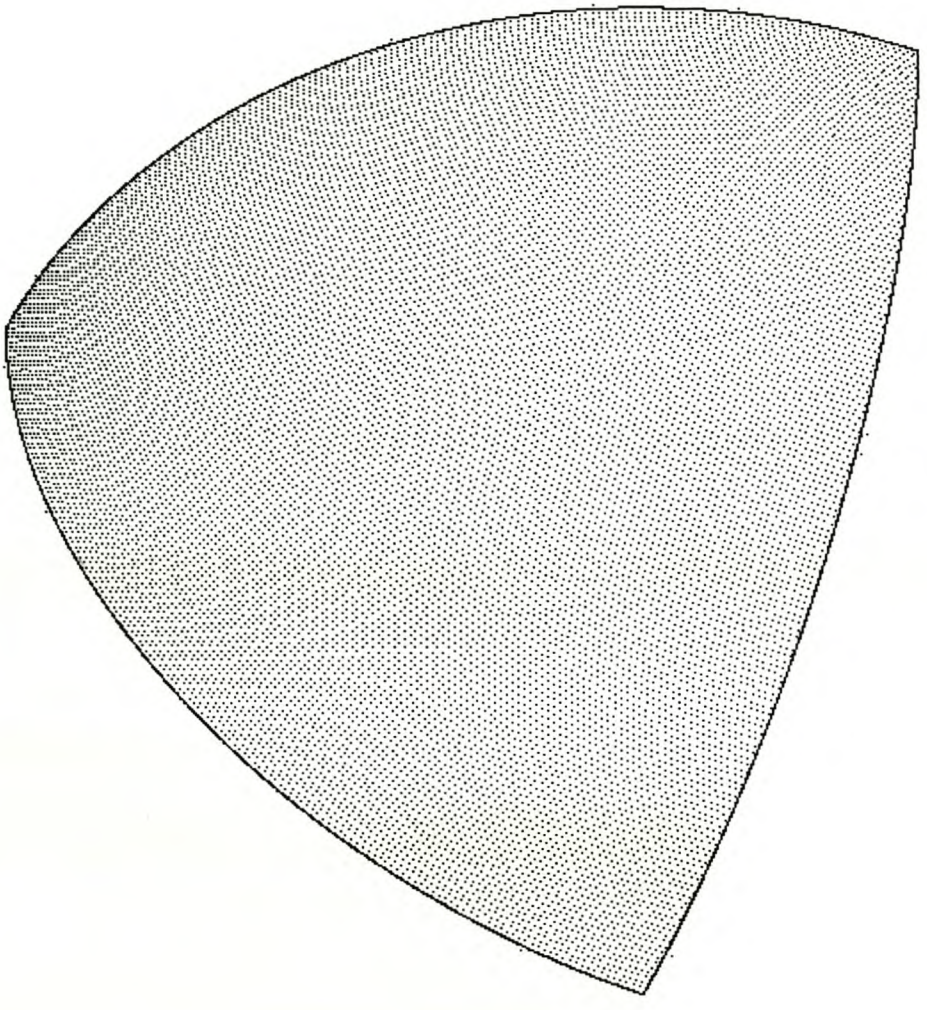

Figure B.62 Extracted sphere number 28

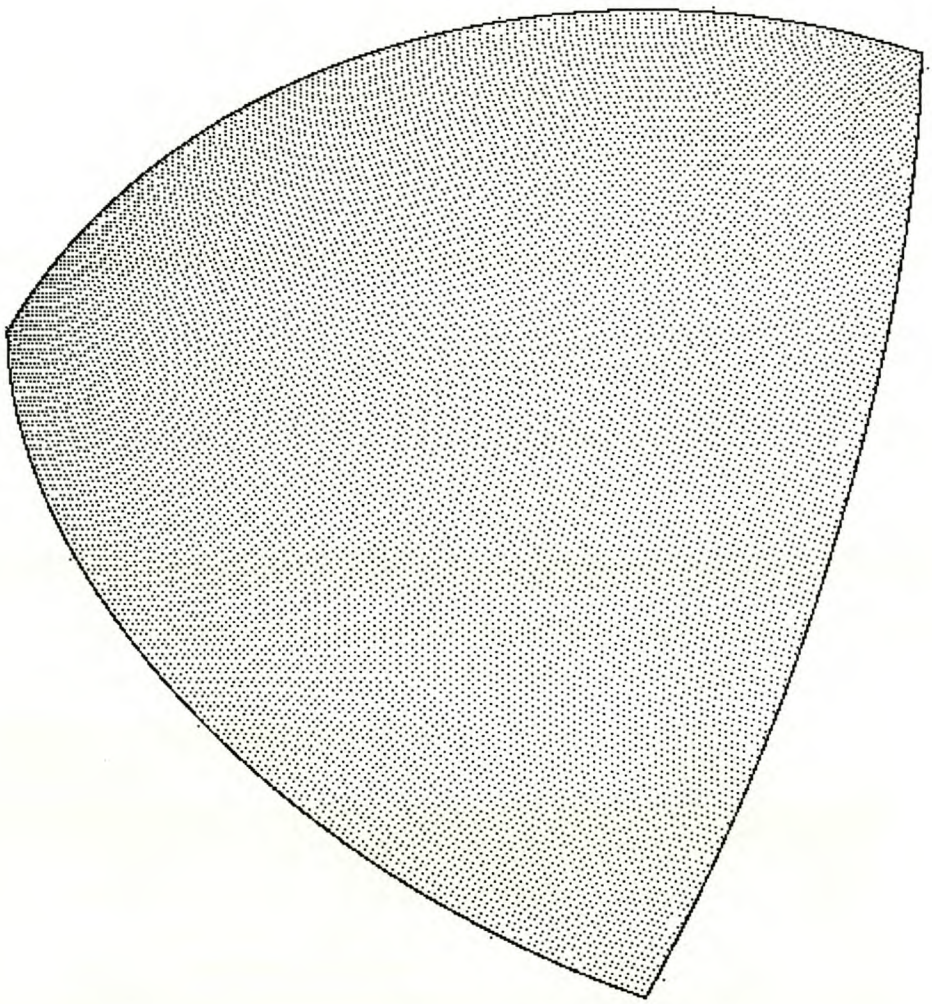

Figure B.63 Extracted sphere number 30

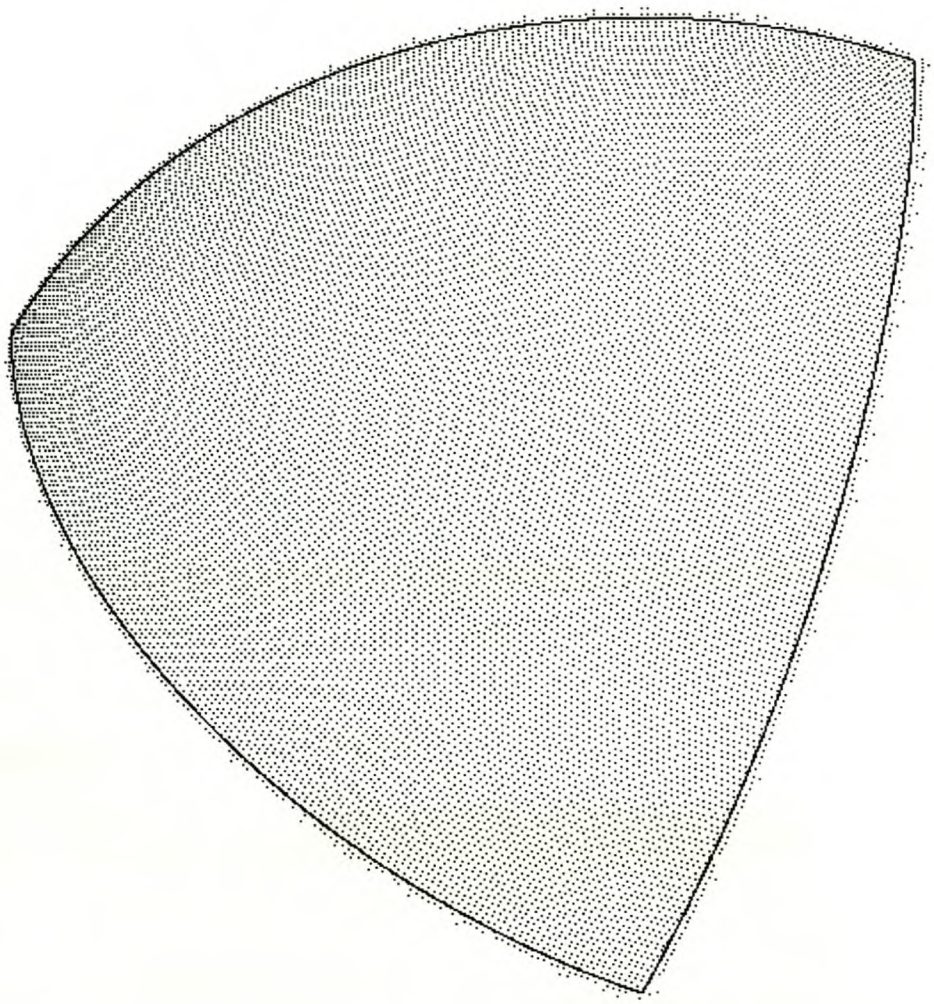

Figure B.64 Extracted sphere number 32

B.3 Cylinders

This paragraph shows extracted small cylindrical disks (20mm radius and 10mm length), extracted small slender cylinders (20mm radius and 80mm length), extracted large cylindrical disks (200mm radius and 100mm length) and extracted large slender cylinders (200mm radius and 800mm length) of Chapter 6. All the well fitted cylinders include some points on the front left and right corners that are included outside the boundary of the cylinder. The reason is that these points lie in the exact cylinder and are one of the properties of the computer generated point cloud chosen. Cylinders showing the same extraction properties are grouped together.

B.3.1 Small Cylindrical Disks

The four cylinders (5, 9, 21 and 25) show the same extraction effect where most of the inner points are included but most of the boundary nodes' points are not included. The reason for this result is the tight boundary node tolerance that is equal to one tenth of the data accuracy.

The two cylinders (1 and 29) show the same extraction effect where most of the inner points are included but points outside the cylinder boundaries are also included. The reason is the large boundary node accuracy (half of the data accuracy) with a data accuracy of 0.05mm.

The two cylinders (13 and 17) can be considered as well fitted with the best one being extracted cylinder 17.

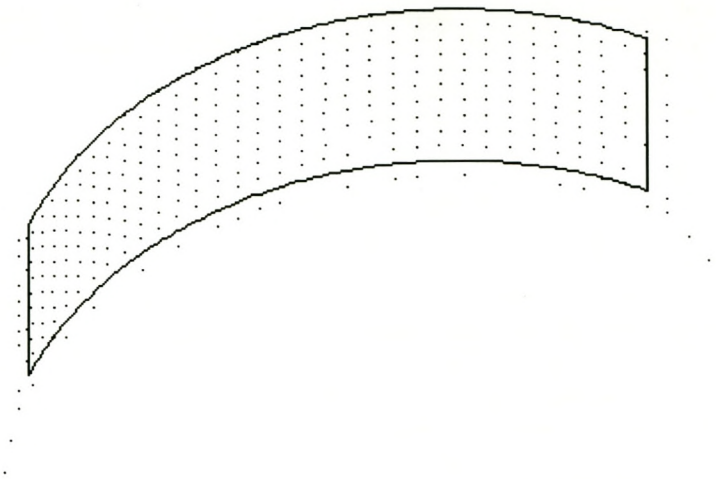

Figure B.65 Extracted cylinder number 1

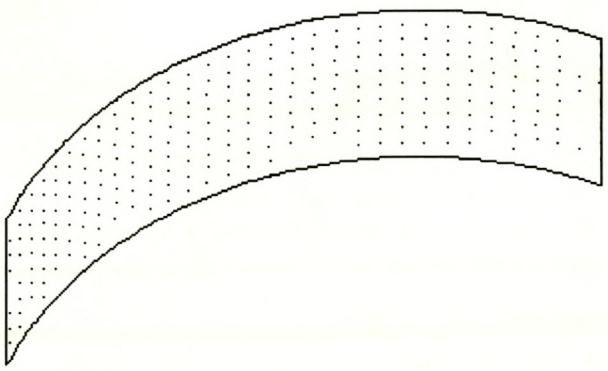

Figure B.66 Extracted cylinder number 5

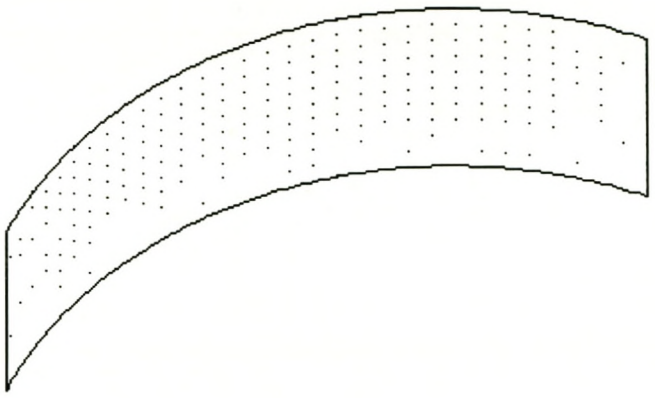

Figure B.67 Extracted cylinder number 9

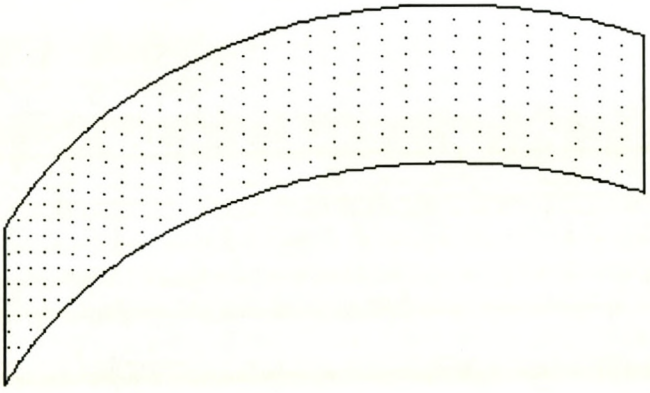

Figure B.68 Extracted cylinder number 13

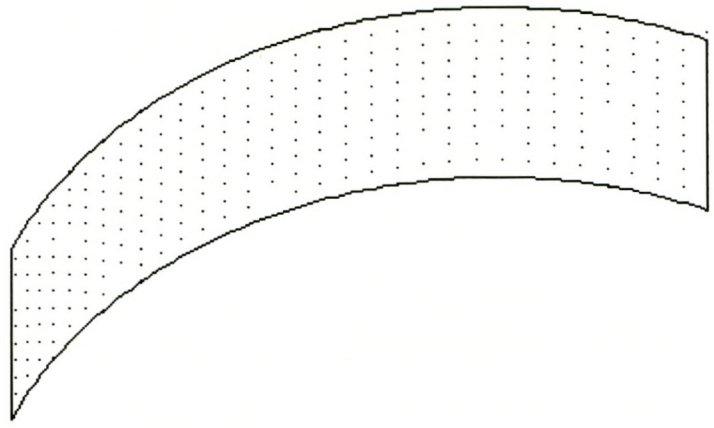

Figure B.69 Extracted cylinder number 17

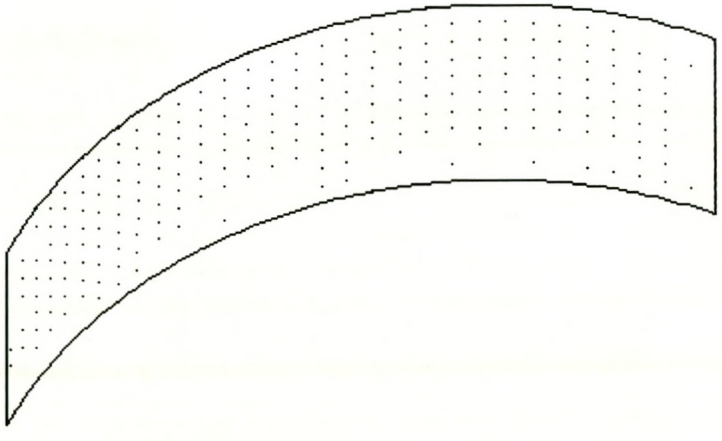

Figure B.70 Extracted cylinder number 21

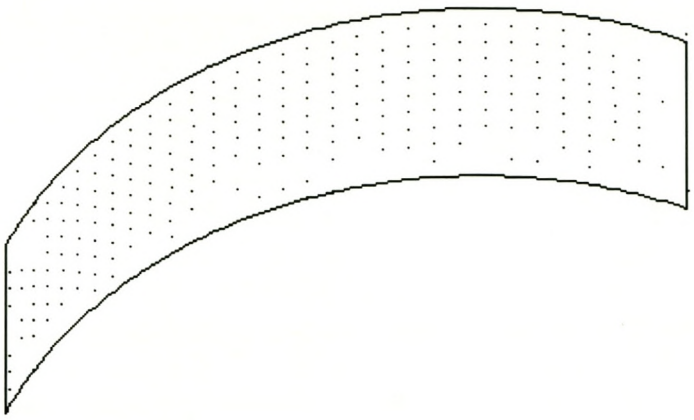

Figure B.71 Extracted cylinder number 25

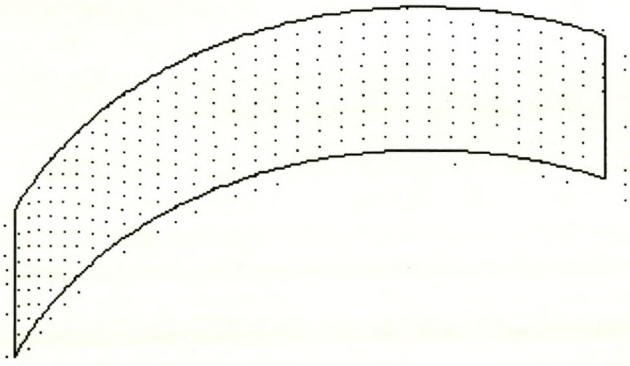

Figure B.72 Extracted cylinder number 29

B.3.2 Small Slender Cylinders

The four cylinders (2, 14, 18 and 30) show the same extraction effect where most of the inner points are included but most of the boundary nodes' points are not included. The reason for this result is the tight boundary node tolerance that is equal to one tenth of the data accuracy.

The four cylinders (6, 10, 22 and 26) can be considered good fitted cylinders with the best one being cylinder number 6 according to the fit parameter values.

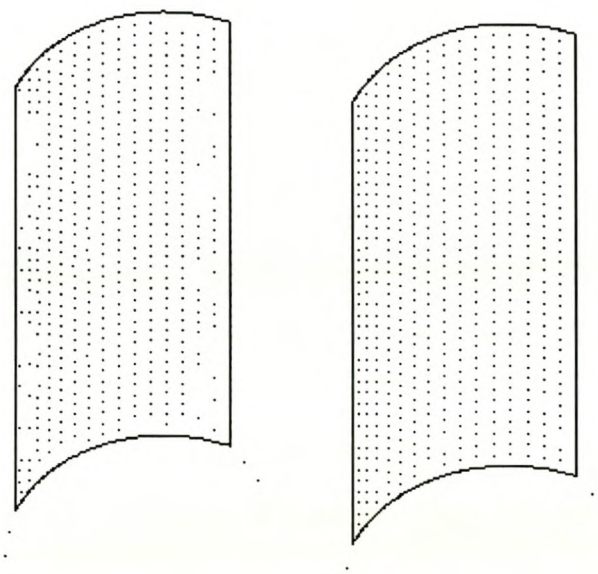

Figure B.73 Extracted cylinders 2 and 6

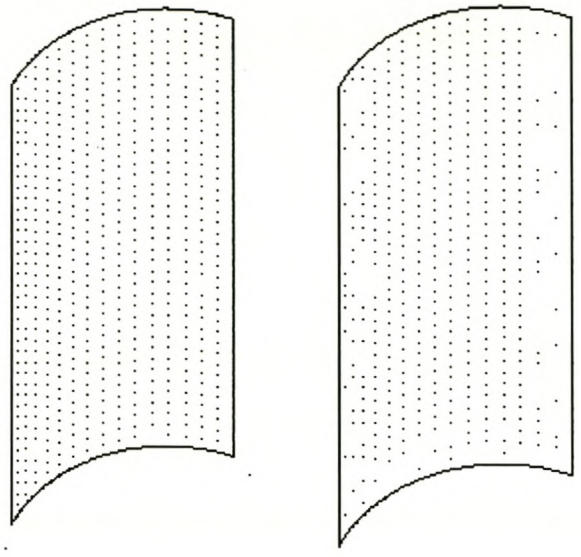

Figure B.74 Extracted cylinders 10 and 14

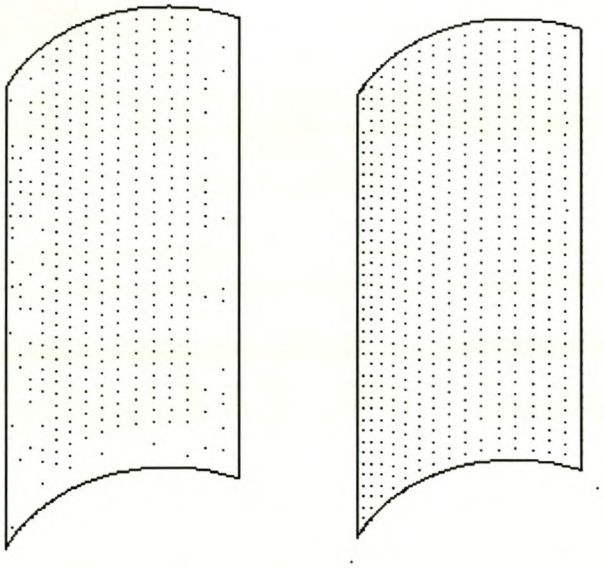

Figure B.75 Extracted cylinders 18 and 22

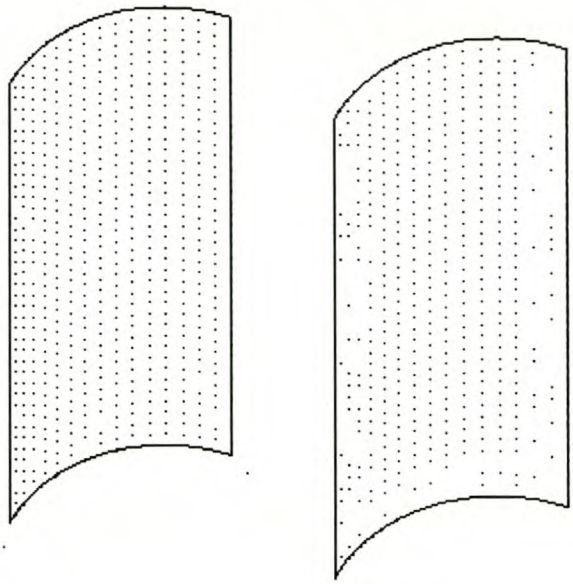

Figure B.76 Extracted cylinders 26 and 30

B.3.3 Large Cylindrical Disks

Only part of the boundary of these cylinders is shown in each case, as the point density is so high that the individual points will not be extinguishable if this entire picture would have been presented. The six cylinders (7, 11, 15, 19, 23 and 27) show the same extraction effect where most of the inner points are included but points outside the cylinder boundaries are also included. The reasons for this result vary from large node sizes (four times the average point pitch), more lenient boundary node accuracies (half of the data accuracy) as well as data accuracies of 0.05mm.

The best fitted cylinders are extracted cylinders 3 and 31. The reader might note the two holes inside each of extracted cylinders 3 (Figure B.85) and 19 (Figure B.86). The reason for this problem is a too tight fit accuracy compared to the node tolerance. Points belonging to these nodes were not included because the nodes were discarded by their node tolerances. A less tight fit accuracy or lower node tolerance solve this problem in both cases.

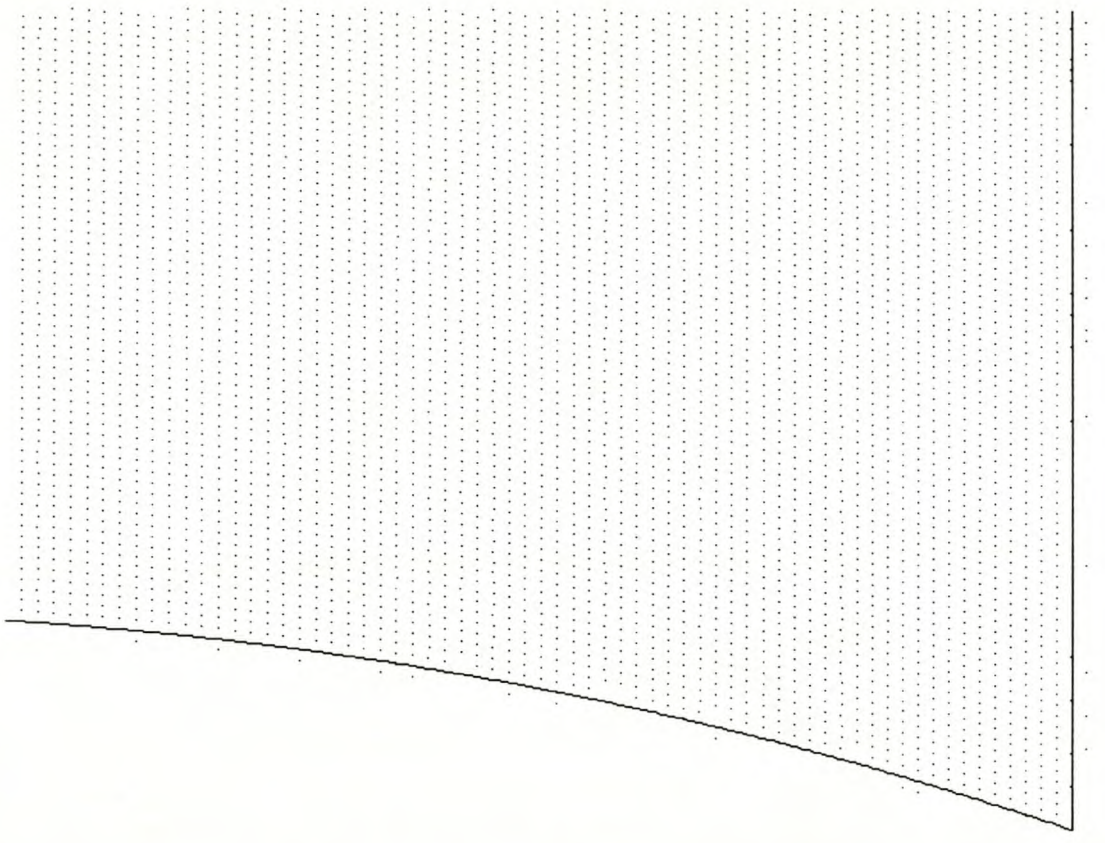

Figure B.77 Part of extracted cylinder number 3

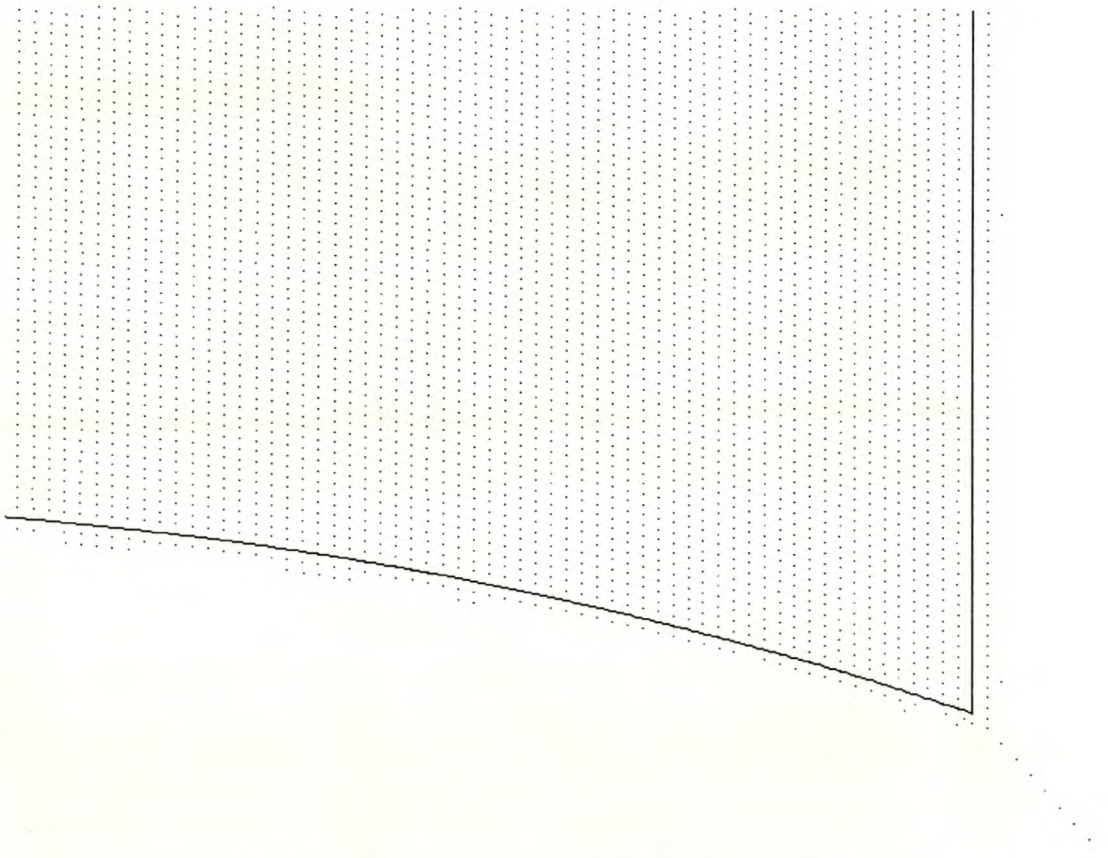

Figure B.78 Part of extracted cylinder number 7

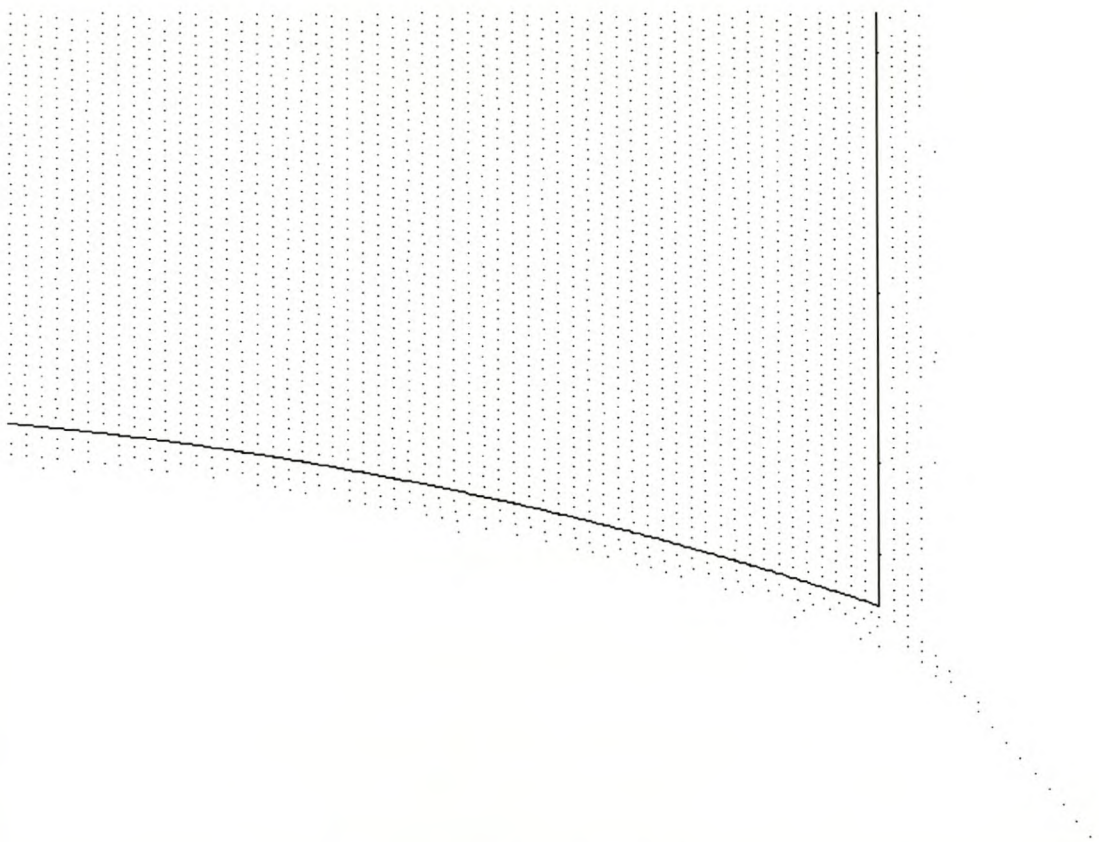

Figure B.79 Part of extracted cylinder number 11

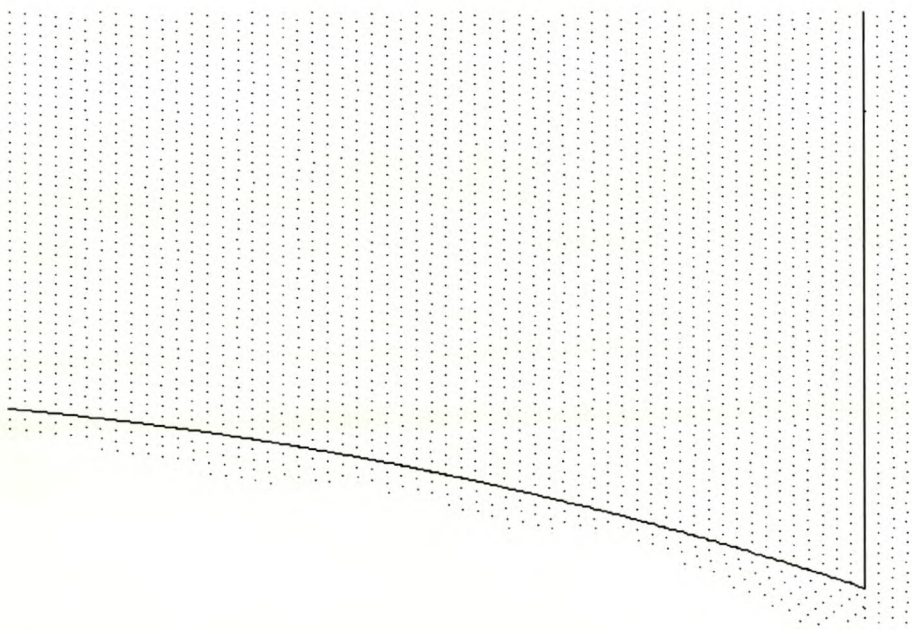

Figure B.80 Part of extracted cylinder number 15

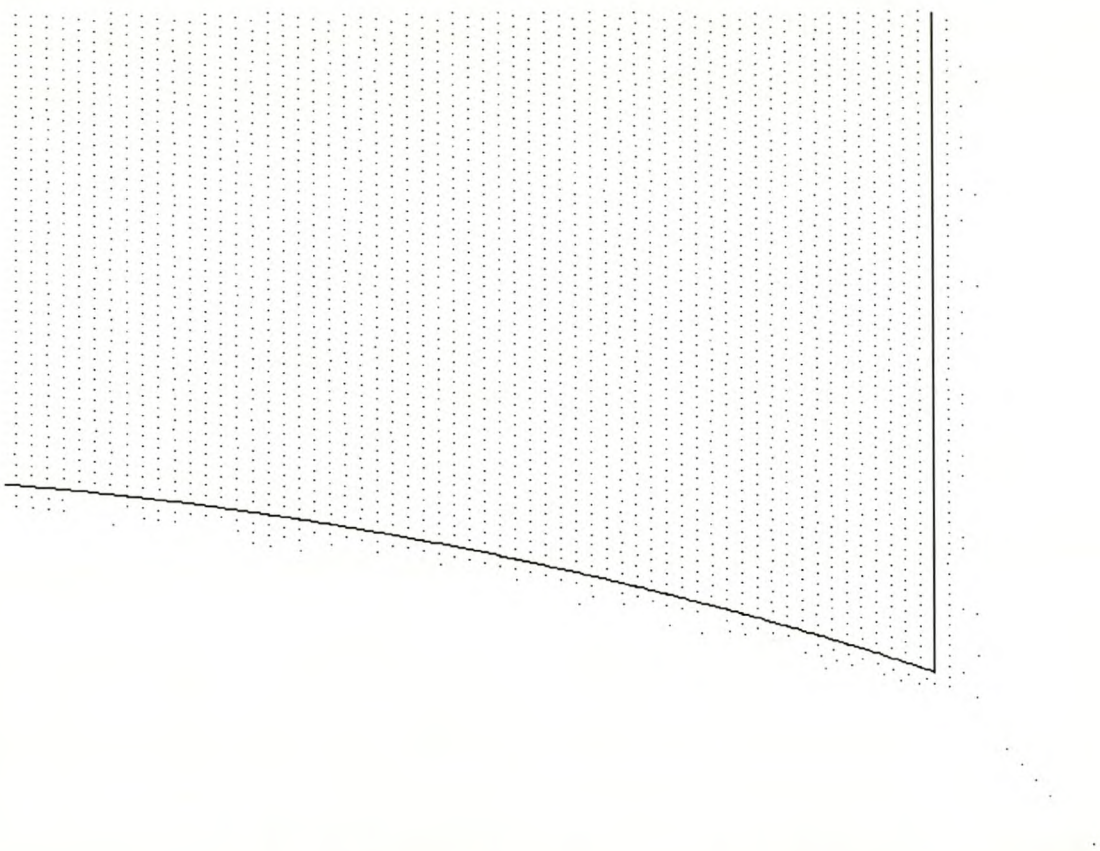

Figure B.81 Part of extracted cylinder number 19

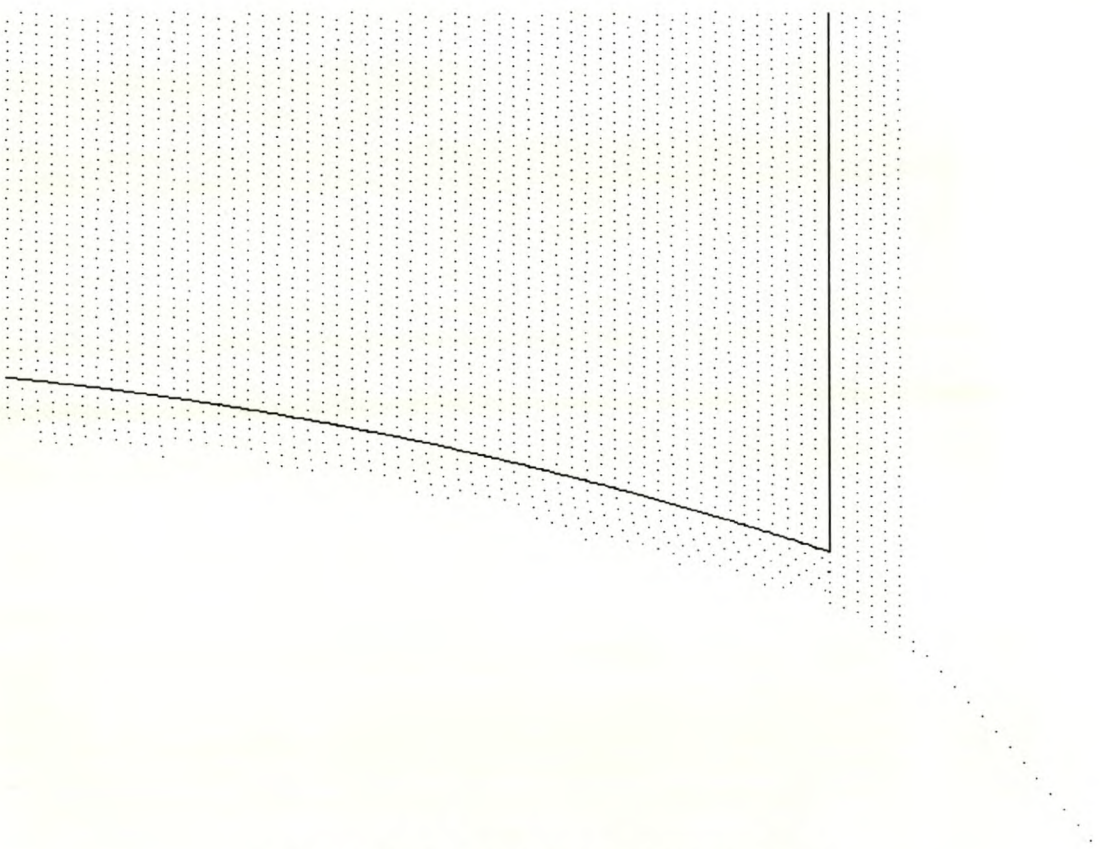

Figure B.82 Extracted cylinder number 23

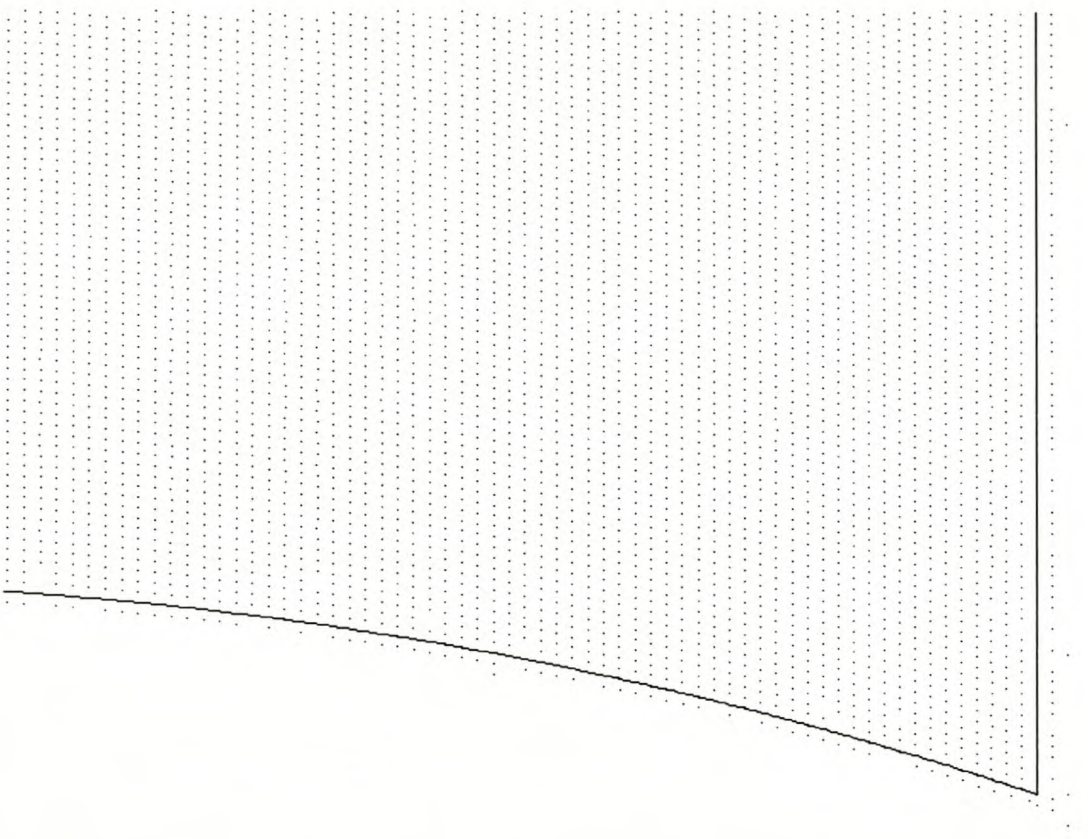

Figure B.83 Part of extracted cylinder number 27

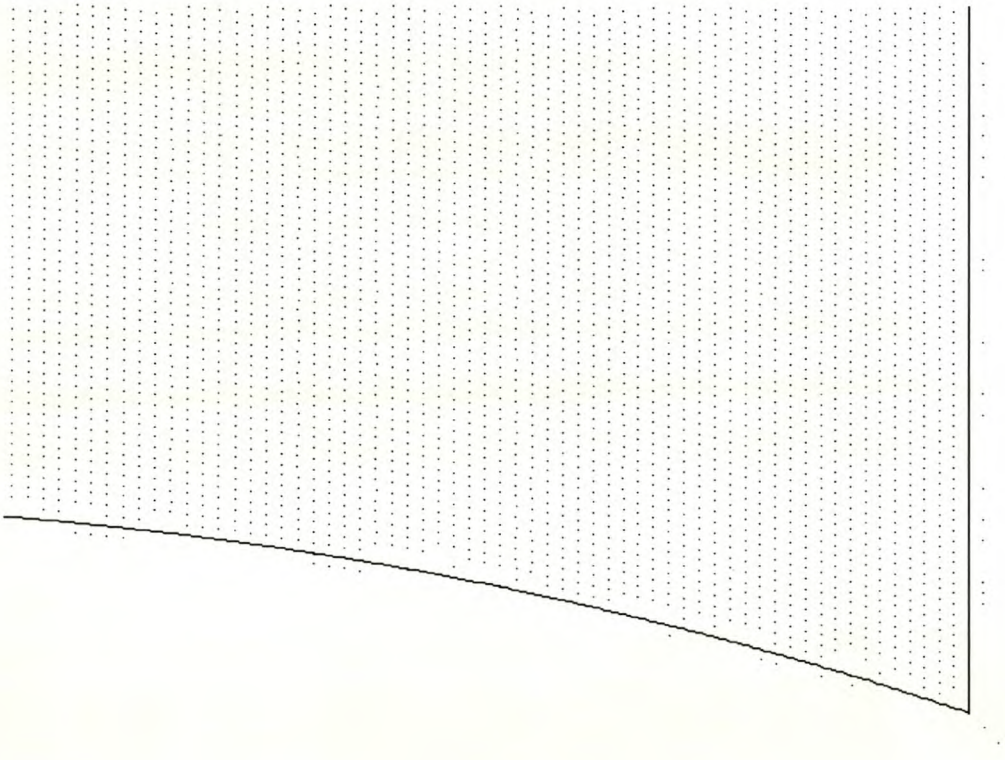

Figure B.84 Part of extracted cylinder number 31

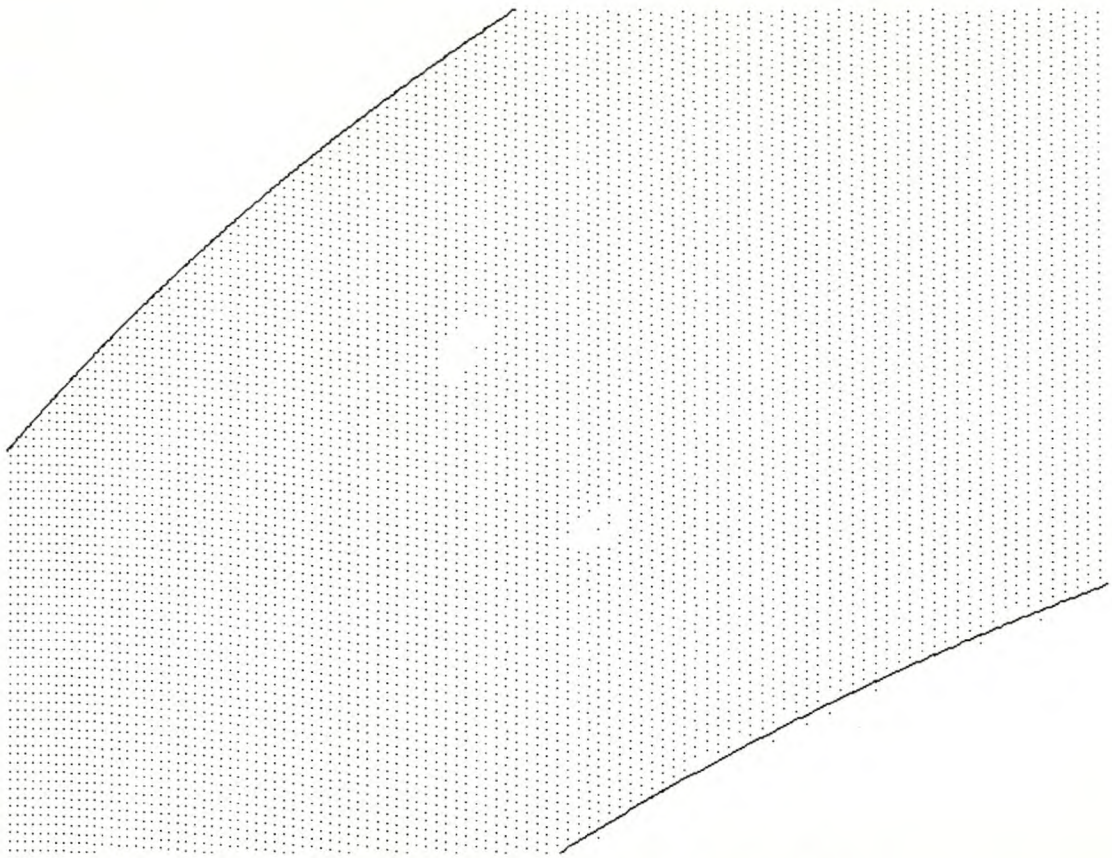

Figure B.85 Two holes of cylinder number 3

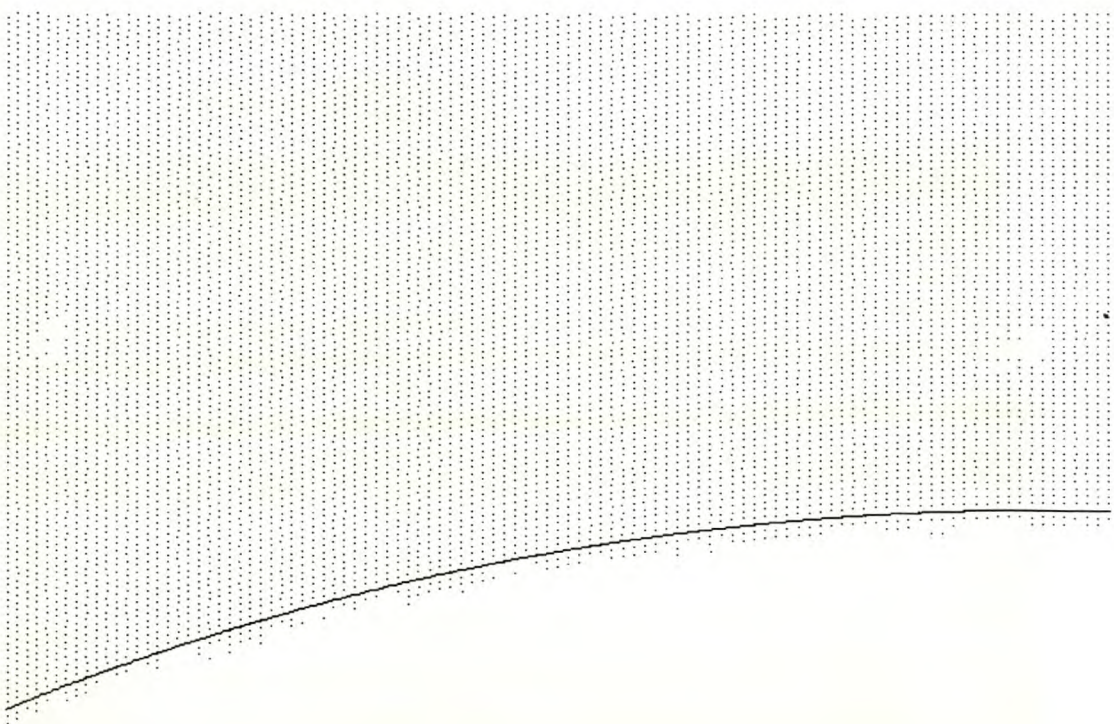

Figure B.86 Two holes of cylinder number 19

B.3.4 Large Slender Cylinders

Only part of the boundary of these cylinders is shown for the same reason as before. The two cylinders (12 and 28) show the same extraction effect where most of the inner points are included but most of the boundary nodes' points are not included. The reason is that the boundary node tolerance is too strict (a tenth of the data accuracy).

The three cylinders (4, 8 and 32) show the same extraction effect where most of the inner points are included but points outside the cylinder boundaries are also included. The reasons vary from large fit accuracies (three times the data accuracy) to large boundary node accuracies (half of the data accuracy equal to 0.05mm).

Extracted cylinders 16, 20 and 24 can be considered as the best ones with cylinder 24 having the best extraction parameter values.

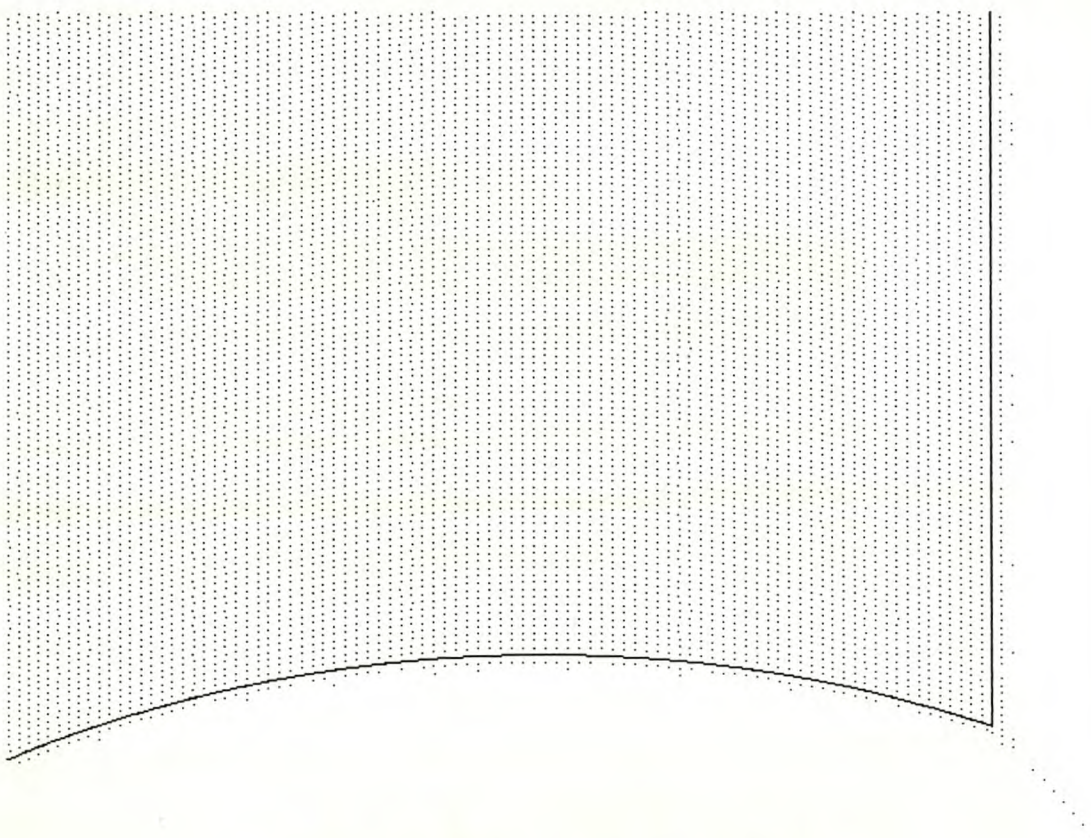

Figure B.87 Extracted cylinder number 4

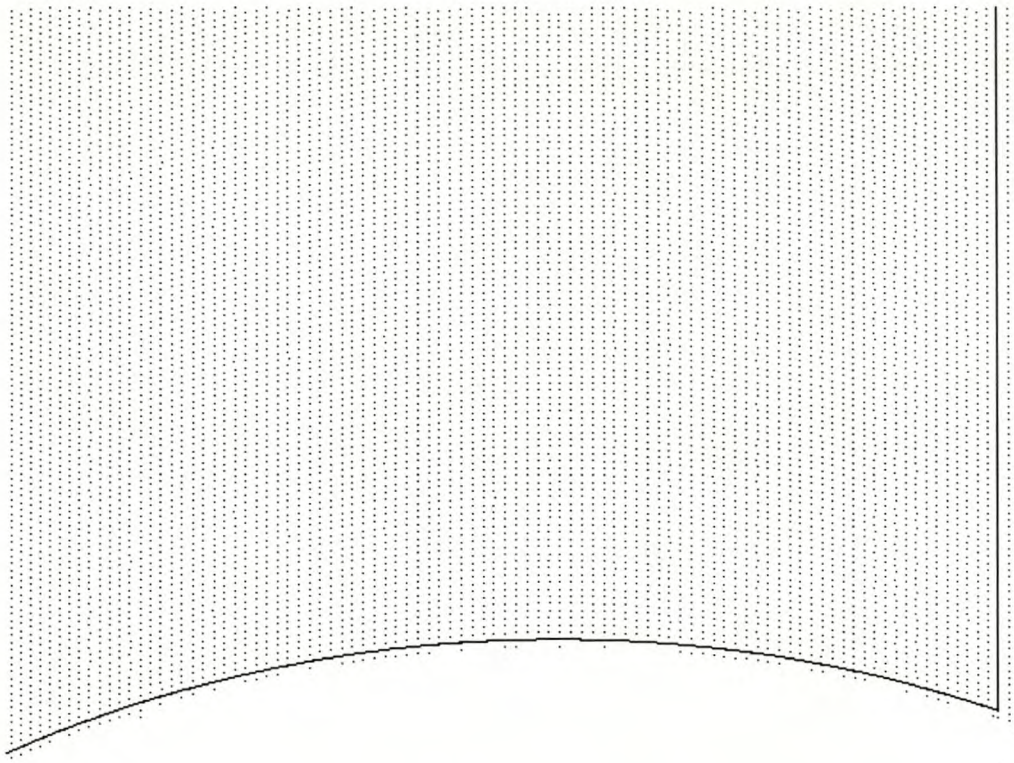

Figure B.88 Extracted cylinder number 8

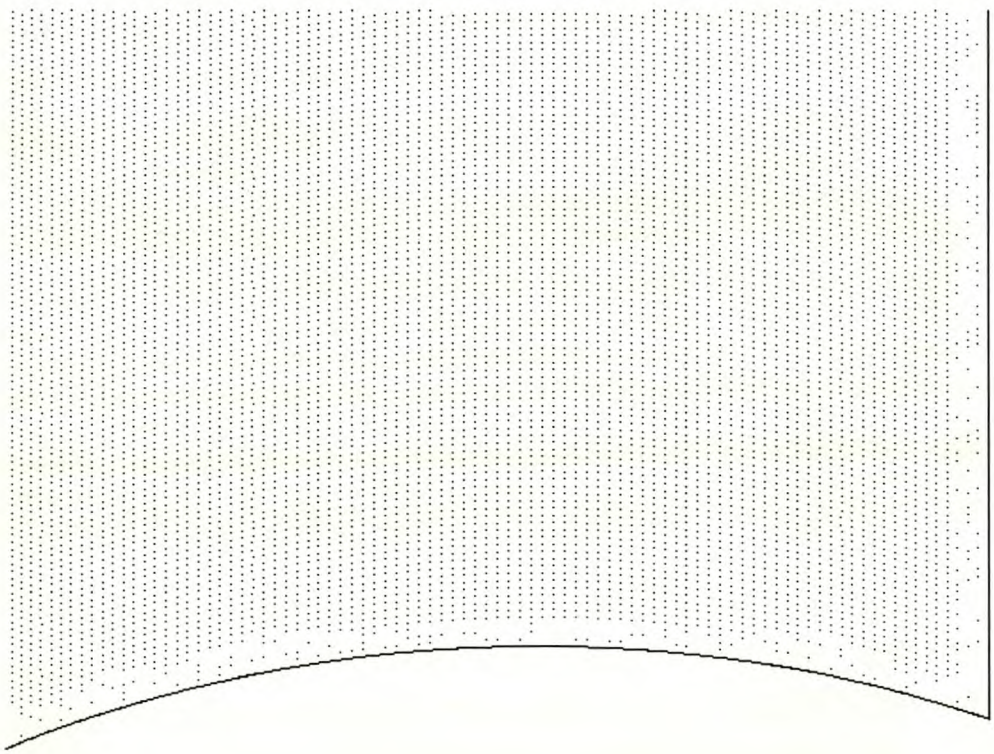

Figure B.89 Extracted cylinder number 12

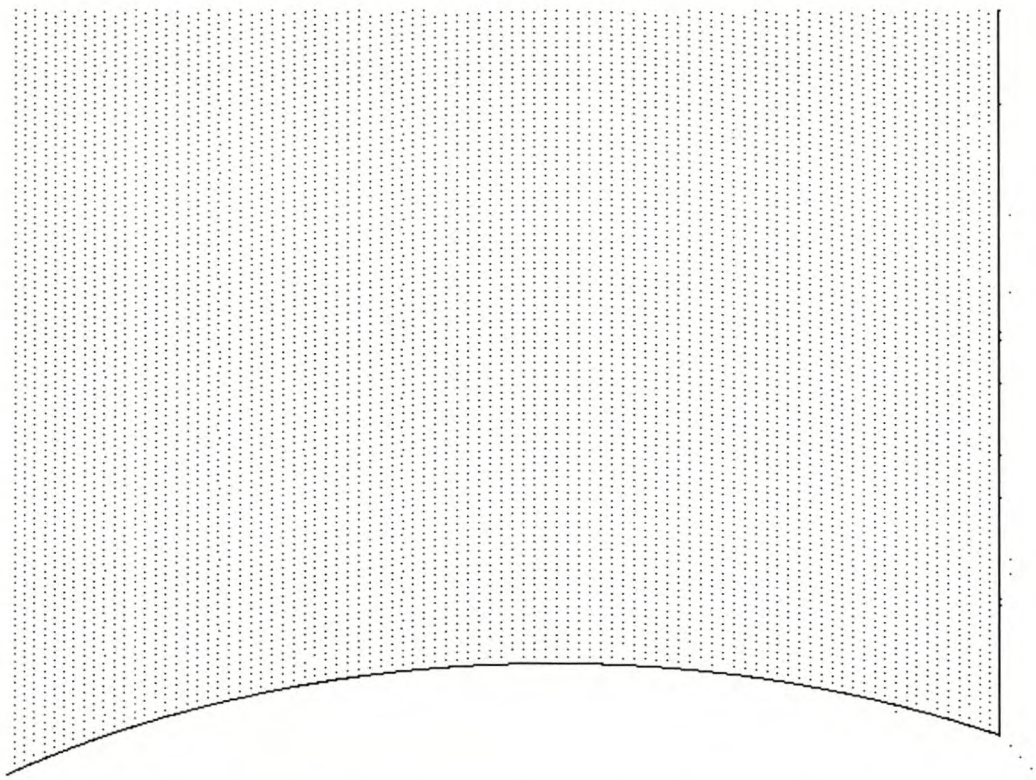

Figure B.90 Extracted cylinder number 16

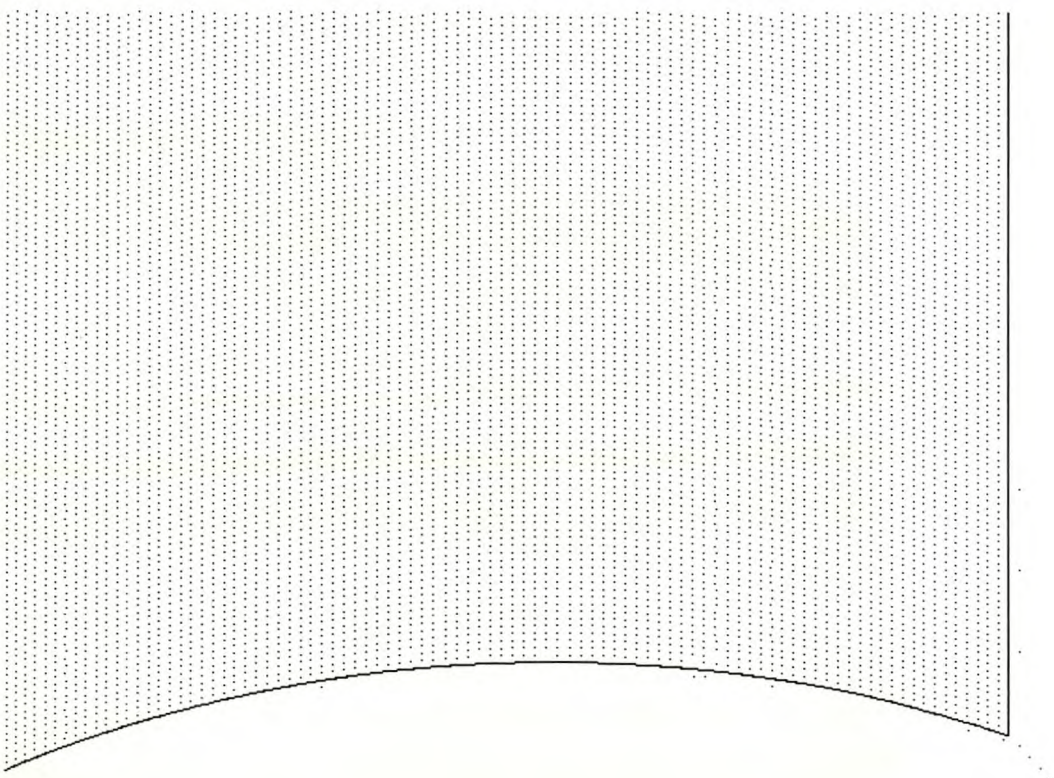

Figure B.91 Extracted cylinder number 20

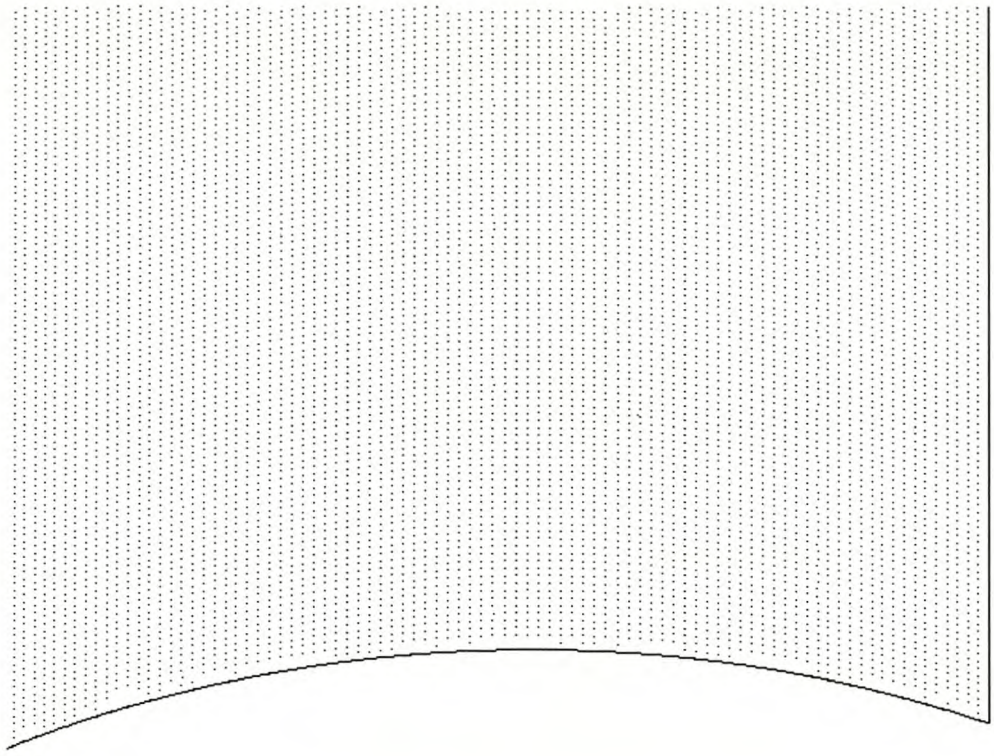

Figure B.92 Extracted cylinder number 24

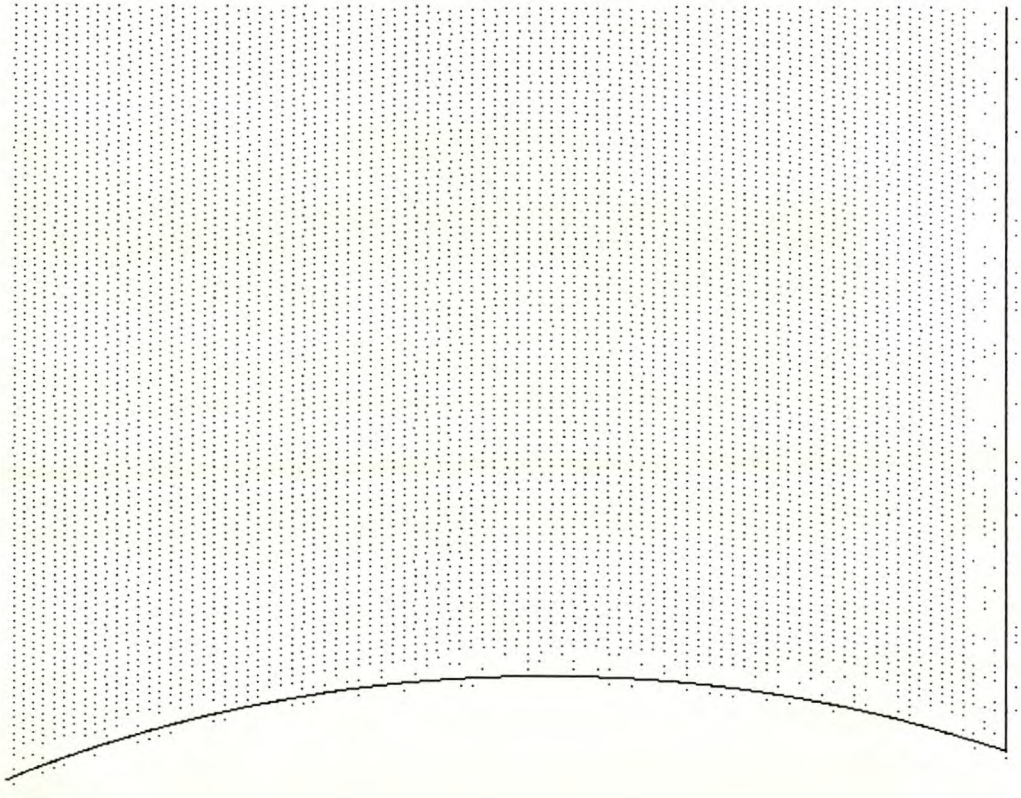

Figure B.93 Extracted cylinder number 28

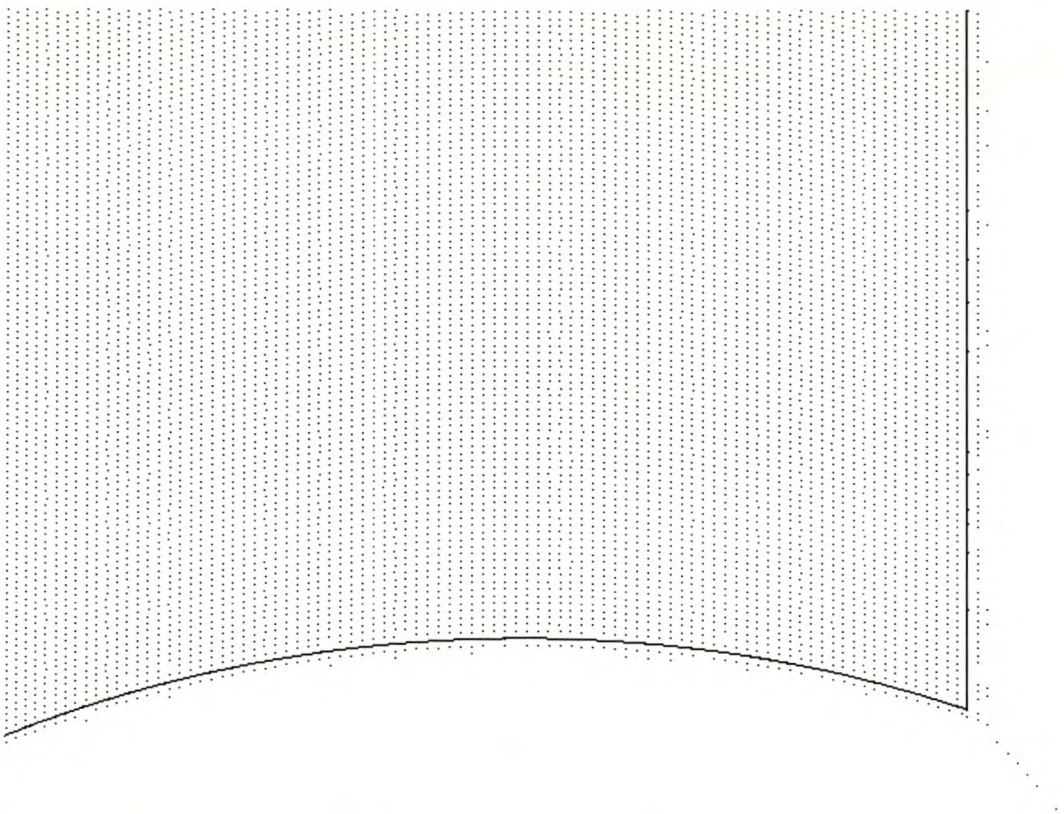

Figure B.94 Extracted cylinder number 32